novatec

Sérgio de Oliveira

INTERNET DAS COISAS
com ESP8266, ARDUINO e RASPBERRY PI

Sérgio de Oliveira

Novatec

À minha filha Elisa e às minhas afilhadas Yasmin e Isadora,
a quem dedico todo o meu carinho.

Sumário

Agradecimentos

Muitas pessoas contribuíram de forma direta e indireta para que este livro fosse publicado. Mesmo com receio de esquecer de alguém que tenha contribuído significativamente, vou lembrar de algumas pessoas.

Para mim, *Internet das Coisas com ESP8266, Arduino e Raspberry Pi* começou nos laboratórios da UFMG. Desde a graduação, quando trabalhamos com sistemas embarcados para PABX e centrais telefônicas, passando pelo mestrado, trabalhando com VoIP em ambiente embarcado WiFi até chegar ao doutorado para atuar em segurança em redes de sensores, tivemos uma longa experiência com as bases de IoT. Pela oportunidade de trilhar esse caminho, gostaria de agradecer a toda a equipe do Departamento de Ciência da Computação da UFMG, em especial aos meus orientadores de graduação, mestrado e doutorado, professores Antônio Otávio Fernandes, Claudionor Coelho Júnior e José Marcos Nogueira. Foram eles que indicaram o caminho a seguir na direção de IoT.

É preciso lembrar também de toda a equipe do Campus Alto Paraopeba da UFSJ, que sempre apoiou e colaborou com este trabalho. Em especial, do grupo de professores da área de Computação Aplicada e dos alunos que foram parceiros na disciplina que deu origem a este livro.

Vale lembrar também de todos os desenvolvedores que desbravaram os conceitos para criar o atual cenário de Internet das Coisas. Todos os softwares e hardwares disponíveis e acessíveis como se encontram nos dias atuais são resultado de um trabalho de muitos anos, boa parte voluntário, de milhares de desenvolvedores que atuam na proposição e construção do hardware, bem como no projeto e no desenvolvimento dos softwares e sistemas, em especial do Linux e do Arduino, ferramentas fundamentais para este trabalho.

Sobre o autor

Sérgio de Oliveira é professor associado do Departamento de Tecnologia do Campus Alto Paraopeba da Universidade Federal de São João del-Rei. Tem graduação, mestrado e doutorado em Ciência da Computação na Universidade Federal de Minas Gerais (UFMG). Realiza pesquisas na área de redes de sensores sem fio, segurança de redes e sistemas embarcados. Atua em convênios e projetos em parceria com empresas e órgãos públicos na área de Internet das Coisas e mobilidade.

Prefácio

Este livro foi escrito durante a oferta de uma disciplina de graduação com o mesmo nome. Os capítulos surgiram de acordo com um cronograma proposto e as demandas dos alunos. A disciplina foi ofertada para alunos dos cursos de Engenharia Mecatrônica e Engenharia de Telecomunicações. No entanto seriam bem-vindos alunos dos cursos da área de Computação e Informática, bem como alunos de outras engenharias. Assim, a sequência de capítulos visa apontar os principais caminhos que possam levar à criação de aplicações de Internet das Coisas.

O principal objetivo deste livro é apresentar como criar aplicações de Internet das Coisas em cenários diversificados. À medida que as aplicações vão surgindo, novos recursos e novas ferramentas vão sendo apresentados e usados, como novas linguagens de programação, módulos e interfaces.

Foram escolhidos módulos de baixo custo, linguagens abertas e ambientes de desenvolvimento gratuitos. A proposta sempre foi desenvolver aplicações de baixíssimo custo, compatíveis com o conceito de Internet das Coisas, visto que Internet das Coisas não pode custar mais do que as“Coisas”que estão se conectando à internet.

Conhecimentos básicos de programação de computadores, sistemas embarcados, internet, redes sem fio, bancos de dados e microeletrônica são importantes para o desenvolvimento de aplicações de Internet das Coisas. Como nem todos os leitores têm esse conhecimento, o livro conta com alguns capítulos introdutórios, visando apresentar os principais conceitos necessários. Assim, com um pouco de esforço e dedicação, entusiastas e curiosos interessados na área podem conseguir entender os conceitos, reproduzir os exemplos e até mesmo propor novas aplicações.

O livro pode ser usado como livro-texto num curso de um semestre em nível técnico ou superior na área. Nesse caso, sugiro ao professor usar a IDE Eclipse customizada para o ambiente Arduino, disponível em *http://eclipse.baeyens.it*, como principal ambiente de desenvolvimento. É

uma ferramenta poderosa, com muitos recursos. O Android Studio também é uma ferramenta importante para o desenvolvimento de interfaces para smartphones Android. De posse dessas ferramentas, muitas novas aplicações podem ser concebidas e desenvolvidas.

PARTE I

Conceitos

CAPÍTULO 1

Introdução

Este capítulo apresenta uma visão geral sobre a área a ser abordada, os recursos existentes e como eles podem solucionar e ajudar nas situações e nos problemas do cotidiano doméstico e profissional.

Internet das Coisas é muito mais que apenas ligar lâmpadas pelo smartphone. Não é somente ligar as “coisas” pela internet, mas também torná-las inteligentes, capazes de coletar e processar informações do ambiente ou das redes às quais estão conectadas. A implantação de Internet das Coisas está mudando totalmente a forma como nos relacionamos com as coisas que estão ao nosso redor, transformando segurança, energia, meio ambiente, trânsito, mobilidade e logística. E o momento atual é propício a essa integração, visto que os dispositivos necessários, agregando as tecnologias para a integração e a conexão, alcançaram preços compatíveis com as “coisas” que se desejam integrar.

O conceito de Internet das Coisas não é novo. Há vinte anos, com a popularização da internet, já se pensava em formas de interligar os equipamentos que usamos no dia a dia com a internet. Algumas tecnologias desenvolvidas nos últimos anos tornaram essa comunicação factível, e com a queda de preços dos elementos envolvidos, em alguns anos, será realidade em todo o mundo. Os módulos baseados no microcontrolador ESP8266 representam um grande avanço na relação de preço-recursos e podem ser um componente muito interessante para soluções de IoT.

A primeira tecnologia associada ao conceito de IoT (Internet of Things – Internet das Coisas) ficou conhecida como RFID (Radio Frequency Identification – Identificação por Radiofrequência). Esta tecnologia surgiu em 1940, com os *transponders* já utilizados nos aviões na Segunda Guerra Mundial. O princípio, simples, continua funcional até hoje. Trata-se de um

equipamento que envia, por radiofrequência, uma identificação única. Para os aviões, sua função é identificar outros aviões ao redor, aumentando a precisão das manobras e dos ataques, além de evitar colisões. Hoje, RFID é usado em crachás, em veículos e até nos produtos em supermercados, substituindo outros tipos de identificação, como código de barras.

As RSSF (Redes de Sensores sem Fio, ou Wireless Sensor Network [WST]) formaram a tecnologia seguinte desenvolvida no âmbito de IoT. São redes compostas de dezenas, centenas ou até milhares de nós microprocessados, com capacidade de comunicação sem fio e alimentados por baterias. Foram propostas especialmente para função de monitoramento, seja ambiental ou em situações de risco, guerra ou emergência. Várias técnicas, diversos protocolos e padrões foram desenvolvidos para atender aos requisitos de redes de sensores, como eletrônica embarcada de baixo custo e consumo mínimo de energia. Pela primeira vez, esperava-se construir um elemento computacional com capacidade de processamento e comunicação sem fio na ordem de grandeza de um dólar. E inúmeras soluções foram desenvolvidas para essa realidade, considerando todas as limitações envolvidas nesse cenário.

De forma paralela, as tecnologias de comunicação e redes de computadores se desenvolveram e popularizaram de forma muito ampla. Primeiro, a internet, usando os protocolos da família TCP/IP. Em seguida, as redes WiFi, que tornaram possível a mobilidade e dispensaram os fios para a interligação de dispositivos próximos. As redes de telefonia celular 2G/3G/4G também foram fundamentais para possibilitar a comunicação de dados dos diversos tipos de equipamentos móveis ou, ainda, aqueles cujo acesso a fios de comunicação eram inviável. A comunicação de dados se tornou acessível a vários tipos de equipamentos e recursos, bem como reduziu o seu custo e o tempo de integração.

Por fim, o desenvolvimento dos dispositivos portáteis, entre os quais os notebooks, tablets e, em especial, os smartphones, trouxe o que faltava para completar o cenário de Internet das Coisas. Com a mobilidade proporcionada por esses equipamentos, ficou muito fácil e desejável acessar as informações e os dispositivos de qualquer parte. A popularização desses dispositivos, em especial dos smartphones, possibilitou que a Internet das Coisas chegasse a ambientes nos quais computadores ainda

são um luxo.

Para tornar Internet das Coisas uma realidade, foi preciso que o custo das soluções se tornasse compatível com as “coisas” a serem conectadas. O microcontrolador ESP8266 da Espressif é uma solução à altura da tarefa, visto que é um circuito totalmente integrado, com interfaces de I/O digitais e analógicas e, ainda, interface WiFi, com um processador de 32 bits, capaz de executar a 160 MHz. Isso tudo ao custo de cerca de 2,5 dólares, considerando já o módulo, pronto para utilização, à venda em sites chineses, para o consumidor final. Com esse módulo, se torna viável colocar um produto de IoT no mercado por 10 dólares, bastante atrativo para um público consumidor bem diversificado.

Este livro apresenta a nova realidade de Internet das Coisas como algo factível e viável para monitoramento e atuação em diversos tipos de cenários e ambientes, desde automação residencial e comercial, passando por sensoriamento ambiental, redes veiculares e trânsito. Serão apresentados os conceitos-base para o funcionamento de Internet das Coisas, o modelo básico de programação e várias aplicações de exemplo. Ele pode ser usado em cursos universitários na área de Computação e Engenharias e também em cursos técnicos especializados nessa área. A única exigência para o entendimento deste livro é o conhecimento básico em programação de computadores e microeletrônica.

A Parte I do livro apresenta os conceitos fundamentais para a área de Internet das Coisas, incluindo os conceitos de Redes de Computadores, Comunicação Sem Fio, Arquitetura de Sistemas Embarcados, Gestão de Energia, Computação na Nuvem e Programação Paralela, Distribuída e de Tempo Real. Esses conceitos são fundamentais para o entendimento dos modelos de desenvolvimento a serem usados nos próximos capítulos.

O desenvolvimento dos sistemas para Internet das Coisas será apresentado em seguida, na Parte II. Os modelos de programação, linguagens e ambientes de desenvolvimento serão apresentados em exemplos simples e didáticos, com enfoque no aprendizado. A linguagem Lua é a primeira linguagem a ser apresentada para o desenvolvimento de aplicações no ambiente NodeMCU, baseado no microcontrolador ESP8266. A seguir, o ambiente Arduino é apresentado usando os módulos que utilizam o

microcontrolador ESP8266. As aplicações com suporte a rede WiFi e a todos os módulos desenvolvidos e suportados no Arduino abrem muitas possibilidades de desenvolvimento de aplicações. Também serão apresentados vários módulos periféricos que podem ser adicionados, com software já disponível para a plataforma Arduino, abrindo várias possibilidades de integração.

A Parte III desde livro apresenta, essencialmente, aplicações de Internet das Coisas, sempre com interface na nuvem. Os diversos dispositivos de IoT se conectam, como clientes, a servidores na nuvem que disponibilizam interfaces, preveem armazenamento e garantem a comunicação com alta disponibilidade. As interfaces para gerenciamento, configuração e relatórios incluem tecnologia web, bancos de dados e aplicativos para smartphones. Também são apresentadas aplicações utilizando as placas Raspberry Pi.

CAPÍTULO 2

Protocolos e padrões

Este capítulo apresenta uma revisão dos conceitos essenciais de redes de computadores, incluindo pilhas de protocolos, padrão WiFi e modelo cliente-servidor.

2.1 Arquitetura cliente-servidor

O modelo de comunicação utilizado na internet é baseado em duas atribuições principais para os elementos envolvidos na comunicação: cliente e servidor. O servidor deve estar sempre disponível, à espera da iniciativa do cliente. O cliente aciona o servidor sempre que precisa fazer a comunicação. Esse princípio difere essencialmente dos modelos de comunicação nos quais qualquer participante da rede pode iniciar a comunicação. No modelo cliente-servidor, a comunicação sempre se inicia no cliente.

O **servidor** é um software que mantém uma porta de comunicação aberta à espera do cliente. Sua localização, seja pelo seu endereço ou nome, deve ser conhecida por todos os clientes que querem acessá-lo. Um servidor pode receber um grande número de solicitações simultâneas de clientes, por isso, normalmente, executa em um computador de alto desempenho. Por causa da demanda de alto desempenho, os computadores de alto desempenho são geralmente denominados também servidores, apesar de o papel de servidor ser desempenhado por um software. Isso não impede que um dispositivo IoT execute a função de servidor, apesar do hardware limitado, recebendo solicitações de clientes.

O **cliente** é também um software, normalmente acionado por um usuário, razão pela qual é comum que tenha uma interface gráfica amigável. Um navegador web como Google Chrome, Mozilla Firefox ou Internet Explorer é um exemplo de cliente. Cabe ao cliente iniciar a comunicação

com o servidor, seja acionada diretamente pelo usuário ou de forma automática, em resposta a um evento ou uma ação externa. Um dispositivo IoT também pode atuar como cliente, acessando servidores para buscar ou atualizar informações sobre seu funcionamento. Como exemplo, um dispositivo de IoT que coleta uma temperatura poderia se conectar a um servidor de bancos de dados periodicamente para enviar as informações de temperatura coletadas, ou quando a temperatura extrapolasse parâmetros previamente definidos.

Diversos servidores podem ser úteis em projetos de IoT, incluindo servidores de arquivos, servidores web, servidores de bancos de dados, servidores de mensagens ou emails, entre outros. Vários deles serão citados ao longo deste livro.

2.2 TCP/IP

Em virtude da complexidade dos sistemas envolvidos, os conceitos de redes de computadores são comumente divididos em camadas que encapsulam um conjunto de funcionalidades e oferecem serviços necessários à conectividade. A ISO (International Organization for Standardization – Organização Internacional de Padronização) propôs o modelo OSI com sete camadas para agrupar as funcionalidades de rede: Aplicação, Apresentação, Sessão, Transporte, Rede, Enlace e Física. Embora haja algumas implementações com o modelo OSI, na prática, o modelo TCP/IP, criado pelo Departamento de Defesa dos Estados Unidos, se tornou muito mais popular. Nesse modelo não existem as camadas de Sessão e Apresentação. Além disso, as funções das Camadas Enlace e Física não estão bem definidas; ficam a cargo de cada implementação. A figura 2.1 apresenta as equivalências entre camadas do modelo OSI/ISO e o modelo TCP/IP.

Modelo OSI/ISO	Modelo TCP/IP
Aplicação	Aplicação
Apresentação	
Sessão	Transporte
Transporte	
Rede	Rede
Enlace	Enlace-Física
Física	

Figura 2.1 – Pilhas de protocolos e suas equivalências.

No modelo TCP/IP, as camadas Aplicação, Transporte e Rede estão bem definidas. E o modelo considera que haja protocolos que façam a entrega dos pacotes na rede local, abaixo da camada de Rede. Para realizar essa função, existem dezenas de protocolos, com destaque para Ethernet, WiFi, PPP (Point-to-Point Protocol – Protocolo Ponto a Ponto) e as redes de celulares 2G/3G/4G, que se tornaram populares com o desenvolvimento da internet.

2.2.1 Camada de aplicação

A camada de aplicação define como os diversos programas vão se comunicar. Isso vai depender muito de cada tipo de aplicação. Algumas aplicações, como fluxo de vídeo em tempo real, precisam manter a conexão durante toda a transmissão do vídeo. Outras, como troca de mensagens de texto, só precisam se conectar periodicamente para verificar se há novas mensagens. Em comum, todas (ou quase todas) as aplicações no modelo TCP/IP usam o modelo de comunicação cliente/servidor.

A princípio, um dispositivo de IoT que esteja ligado a um equipamento qualquer tem características de uma aplicação servidor, pois está sempre pronto para receber solicitações de acionamentos ou das informações por ele disponibilizadas. Na prática, entretanto, a confiabilidade do serviço pode ficar comprometida nessa configuração em virtude da confiabilidade da rede, dos roteadores e dos demais equipamentos presentes nesse cenário. É difícil garantir até a disponibilidade de energia para os

dispositivos IoT, que podem estar em ambientes instáveis e hostis.

Para garantir mais confiabilidade aos serviços prestados pelos dispositivos IoT é interessante considerar a presença de um equipamento intermediário, disposto em um ambiente de maior confiabilidade, que faça a interface entre as solicitações recebidas pelos usuários e as informações disponibilizadas pelos dispositivos IoT. O protocolo MQTT (Message Queue Telemetry Transport) atende a essa demanda por meio de equipamentos chamados *brokers*, que fazem essa intermediação. Há vários brokers disponíveis na internet de forma gratuita. Nessa configuração, associa-se o conceito de IoT ao conceito de *Cloud Computing* (Computação na nuvem), visto que o broker está na "nuvem", ou seja, em algum lugar da internet cuja localização não é importante, apenas o serviço que ele disponibiliza.

2.2.2 Camada de Transporte

A camada de transporte tem algumas funções essenciais ao funcionamento da rede. Seu principal objetivo é entregar o fluxo de dados às aplicações. Assim, tem uma primeira função de chavear os diversos pacotes de dados que chegam ao computador para as diversas aplicações que fazem acesso à rede. Além disso, pode agregar outras funções como controle de fluxo e congestionamento, garantia de entrega dos pacotes, incluindo a retransmissão de pacotes perdidos e o reordenamento de pacotes.

Para realizar o chaveamento dos pacotes pelas aplicações, a camada de Transporte usa um número entre 0 e 65.535 (16 bits) como referência. Esse número pode ser considerado o endereço de transporte e também é chamado de porta de transporte. Todo fluxo de comunicação tem um endereço de transporte de origem, indicando qual a aplicação é responsável pelos pacotes na origem, e um endereço de transporte de destino, indicando a aplicação correspondente no destino.

Quando uma aplicação Cliente inicia a comunicação com uma aplicação Servidor, ela deve indicar qual é o endereço de transporte da aplicação Servidor que se quer acessar. A maioria das aplicações usa endereços fixos para facilitar seu acesso, evitando, assim, que se indique o endereço de transporte e tornando seu acesso mais transparente. É assim que as

aplicações reservaram seus endereços bem conhecidos: Web (80), DNS (53), SMTP (25), SSH (22), Telnet (23), FTP (20 e 21), entre outros.

O protocolo de transporte mais utilizado é o TCP (Transmission Control Protocol), mas não é o único. O protocolo UDP (User Datagram Protocol) é usado em aplicações que não requerem garantia de entrega, reordenamento de pacotes ou qualquer controle de fluxo ou congestionamento. Basicamente, o protocolo UDP tem a função exclusiva de chavear os pacotes para as aplicações. Há, ainda, o protocolo ICMP (Internet Control Message Protocol), usado apenas em aplicações de teste, como o `ping`, aplicação famosa para verificar se um computador está ativo e acessível.

2.2.3 Camada de rede

A camada de rede é responsável pelo endereçamento universal. Ou seja, ela deve definir endereços de origem e destino dos pacotes que, a princípio, não deveriam ser alterados até seu destino. O protocolo usado nessa camada é o protocolo IP (Internet Protocol). Atualmente, duas versões desse protocolo convivem: a versão 4 e a versão 6. A versão 4 tem como principal problema o número limitado de endereços disponíveis, visto que usa apenas 32 bits para endereçamento, problema que é resolvido na versão 6, na qual o endereçamento passa a ter 128 bits.

Os 32 bits de endereçamento do IPv4 suportam o máximo teórico de 4 bilhões de elementos conectados, número que já seria insuficiente em um cenário de IoT mundial. Além disso, os endereços estão mal distribuídos pelo mundo, sendo muito mais escassos em regiões que demoraram a aderir à internet. Devem ser considerados, ainda, vários endereços reservados que não podem ser usados por ter funções especiais. Com 128 bits do IPv6, seria possível endereçar cerca de 1039 dispositivos, o que representa bilhões de bilhões de bilhões de bilhões, ou seja, mais que suficiente para a realidade que se consegue imaginar para IoT.

Para facilitar a memorização e a configuração das redes IPv4, os endereços de 32 bits são normalmente agrupados em 4 octetos separados por ponto, cada qual pode receber um valor entre 0 e 255. Um endereço IPv4 pode ser, ainda, dividido em duas partes: endereço da sub-rede e endereço do

elemento na sub-rede. O número de bits destinado à sub-rede é variável, sendo indicado por um outro valor chamado **máscara de sub-rede**. A máscara pode ser indicada por um número que informa quantos bits são usados para a sub-rede, ou por um conjunto de 32 bits, também agrupados em 4 octetos separados por ponto, em que os bits para a sub-rede recebem o valor igual a 1 e os bits para o elemento recebem o valor igual a zero. Há uma lógica associada à essa representação: o operador E (AND) lógico aplicado entre o endereço IP e a máscara resulta no endereço da sub-rede.

Exemplo:

Endereço IP: 200.125.23.44

Máscara: 255.255.255.0 (ou apenas 24)

Endereço da sub-rede: 200.125.23.0

A configuração dos computadores nas redes também exige a indicação de um **gateway**, que se trata do computador de saída da rede, sua interface com a internet, normalmente indicada pelo endereço IP do roteador. Para o correto funcionamento da rede, os endereços de todos os computadores de uma sub-rede, bem como o endereço do roteador/gateway, devem ter endereços IP definidos na mesma sub-rede, ou seja, a operação AND lógica dos endereços com a máscara de sub-rede deve ter como resultado o mesmo endereço de sub-rede.

2.2.4 DHCP e DNS

A tarefa de configuração manual das redes deve ser automatizada sempre que possível, visando minimizar o trabalho dos administradores de rede e agilizando o processo de entrada de acesso à rede. Para atender a esse objetivo, o protocolo **DHCP** (Dynamic Host Configuration Protocol – protocolo dinâmico de configuração de computadores) foi difundido de forma ampla e hoje está disponível em todos os computadores e roteadores que suportam o protocolo IP.

O protocolo DHCP funciona no modelo cliente-servidor. Um equipamento de rede, normalmente o roteador que atua como gateway da rede, recebe a função de servidor e permanece constantemente ligado à rede para receber solicitações de configuração dos demais computadores. Assim que um

computador é ligado à rede, ele envia uma mensagem à rede solicitando sua configuração. O servidor recebe essa mensagem e a responde, enviando uma configuração disponível para ser utilizada. Para que os computadores enviem essa mensagem, eles devem ser configurados com a opção "obter um endereço IP automaticamente", ou algo parecido.

Além de receber o endereço IP, o computador recebe também a máscara de sub-rede, o endereço do gateway e, possivelmente, o endereço dos computadores de rede que fazem o serviço de tradução de nomes para IP, conhecido como DNS.

O serviço **DNS** (Domain Name System – sistema de nomes de domínio) é responsável por traduzir nomes que são mais fáceis de memorizar, para endereços IP. Assim, se torna desnecessário memorizar os endereços IP de todos os computadores que se queiram acessar. O acesso pode ser feito por meio de um nome que pode estar diretamente ligado ao serviço que se queira acessar. Assim, nomes como *google.com*, *www.ufsj.edu.br*, ou qualquer outro, podem ser localizados na internet. Para que um computador consiga acessar serviços pelo nome, é preciso que o nome esteja devidamente registrado e que um servidor de DNS esteja disponível e configurado.

Além da configuração de rede, é possível também carregar sistemas operacionais ou firmwares pela rede, durante a inicialização dos sistemas, por um serviço similar ao DHCP, conhecido como **TFTP** (Trivial File Transfer Protocol). Esse serviço pode ser interessante em IoT, pois possibilita que o software de toda a rede esteja centralizado em um único elemento, sendo facilmente atualizado e carregado, além de, possivelmente, reduzir o custo dos dispositivos IoT, se eles não contarem com memória para armazenamento dos seus programas. A placa Raspberry Pi, que será usada em alguns exemplos deste livro, pode ser inicializada pela rede, reduzindo, assim, o tamanho mínimo do cartão de memória SD, necessário para seu funcionamento.

2.2.5 NAT (Network Address Translation)

A migração do IPv4 para o IPv6 tem sido mais lenta do que se esperava. Entre os motivos estão os problemas de compatibilidade e reconfiguração

das redes. Além disso, soluções de mapeamento de endereços estão resolvendo, ao menos parcialmente, os problemas de falta de endereços. A principal solução de mapeamento de endereços falsos, conhecida como NAT (Network Address Translation), se tornou extremamente popular e está disponível na totalidade dos equipamentos de rede.

Pelo NAT é possível utilizar apenas um endereço IP válido para toda uma sub-rede. Pelo mecanismo, um único endereço válido é associado ao roteador de saída, que faz a tradução de todos os endereços da sub-rede interna que não são válidos na Internet para o único endereço válido. No retorno dos pacotes, a tradução é refeita no sentido contrário. A referência sobre quais pacotes correspondem a cada endereço falso utilizado é realizada pelo endereço da porta de origem da camada de transporte, informação que seria pouco relevante nesse processo e que pode ser alterada caso dois endereços de porta de origem coincidam para dois endereços IP falsos distintos. A figura 2.2 ilustra a tradução de endereços IP falsos para um endereço IP válido.

Figura 2.2 – Tradução de endereços para IP falsos.

No retorno da comunicação, ao receber a resposta, o roteador segue o mesmo mapeamento para converter os endereços, devolvendo os pacotes aos seus destinos correspondentes.

O protocolo NAT funciona bem quando a requisição é enviada por um computador que tem um IP falso. Contudo, se o servidor também tiver um IP falso, em outra rede, o NAT poderá inviabilizar a comunicação, visto que o cliente só pode gerar uma requisição para um IP válido na rede. Isso poderia gerar um problema para os dispositivos IoT caso eles disponibilizassem alguma aplicação servidor. Como foi apresentado, as aplicações do tipo Servidor devem ter um endereço conhecido para ser acessado pela aplicação Cliente. Para resolver esse problema, o protocolo NAT prevê a possibilidade de fixar um mapeamento pela porta de transporte. Assim, por exemplo, é possível definir que o roteador mapeie sempre os pacotes recebidos com uma determinada porta de destino, para um determinado endereço IP falso, previamente configurado.

A configuração do protocolo NAT para mapeamento fixo pela porta resolve o problema de acessar aplicação servidor no dispositivo IoT, mas gera outro problema: configurar o roteador. Existem centenas de fabricantes e tipos de roteadores diferentes, e cada um tem uma interface diferente para essa configuração. Exigir que o usuário configure seu roteador para cada dispositivo IoT presente em sua rede se torna inviável em consequência do número de dispositivos IoT que se espera conectar. Para resolver esse novo problema foi proposto um novo protocolo: UPnP.

O protocolo UPnP atua na configuração automática de roteadores e dispositivos de rede que utilizem o protocolo NAT para mapeamento de endereços falsos em um IP válido. Ele é iniciado quando o dispositivo é ligado na rede, e o processo pode ser repetido periodicamente para que a configuração seja refeita, mesmo que o roteador seja reiniciado ou perca sua configuração por algum motivo. Os roteadores e os demais equipamentos de rede que suportem esse protocolo são capazes de identificar as solicitações e, automaticamente, configurar o mapeamento da porta solicitada para o endereço IP que esteja solicitando o mapeamento. As portas são distribuídas de forma automática à medida que os vários dispositivos vão se conectando à rede.

2.3 WiFi

O padrão IEEE 802.11 ficou internacionalmente conhecido e popularizado

como WiFi, acrônimo para *Wireless Fidelity*, em alusão à *HiFi*, termo usado para amplificadores de áudio de alta qualidade. Sua proposta, de conectar dispositivos em redes locais sem fio, sempre foi questionável em vários aspectos, mas isso não impediu que esse padrão se tornasse tão popular quanto a internet. Apresenta várias versões (a/b/g/n/ac) e tem, ainda, muitos problemas a ser solucionados para que não seja substituído por outra tecnologia que atenda melhor à evolução de Internet das Coisas.

As redes WiFi consideram um número finito de elementos de rede ligados em um mesmo canal, porém não há limite teórico para esse número. É comum a presença de um elemento de rede dedicado a interligar a rede WiFi à rede cabeada provendo seu acesso à internet. Esse elemento dedicado é conhecido como AP (Access Point – Ponto de Acesso) ou roteador wireless.

Na pilha de protocolos TCP/IP, o padrão WiFi ocupa a parte de baixo, respondendo pelas funções de subcamada física, acesso ao meio e ligação de dados. Em princípio, os dispositivos conectados à uma mesma rede WiFi local poderiam se conectar diretamente, sem a necessidade das demais camadas TCP/IP, mas, na prática, não existem implementações populares de protocolos de comunicação WiFi sem a pilha TCP/IP, o que justifica considerar WiFi totalmente integrado ao TCP/IP.

2.3.1 Subcamada física

Na parte física, o padrão WiFi prevê dois conjuntos de frequências: 2,4 GHz e 5,8 GHz. A faixa de 2,4 GHz vai de 2,40 a 2,4835 GHz, na qual estão distribuídos 11 canais. No Brasil é permitido também o uso dos canais 12 (2.467 GHz) e 13 (2.472 GHz), assim como na maior parte dos países da Europa. Entretanto a maioria dos equipamentos opera apenas dentro dos 11 canais permitidos nos Estados Unidos, o que inviabiliza os canais 12 e 13.

Os canais WiFi foram distribuídos com um problema muito grave. Eles estão distribuídos com apenas 5 MHz de intervalo, mas, na realidade, utilizam 22 MHz de largura. Com isso, afetam os canais laterais, de forma a inviabilizar sua utilização simultânea. Um canal central ocupa a faixa de 7 canais, 3 abaixo e 3 acima, além do próprio canal. Com isso, a melhor

utilização de WiFi é usar somente os canais 1, 6 e 11 e isolar todos os outros canais.

Um aplicativo disponível para Android, WiFi Analyzer (*http://wifianalyzer.mobi*), indica os canais ocupados pelas redes acessíveis ao equipamento no qual está instalado. A figura 2.3 mostra a tela do aplicativo WiFi Analyzer. É possível ver claramente a sobreposição dos canais.

A faixa de frequência de 5,8 GHz é muito mais ampla, vai de 5,725 a 5,85 GHz e tem 23 canais que não se sobrepõem. Porém os equipamentos para trabalhar em frequência mais alta são mais caros. Normalmente, os dispositivos IoT não suportam essa frequência. Entre os padrões IEEE 802.11, o padrão ‘b’ e o ‘g’ não suportam essa frequência. Os padrões ‘a’, ‘n’ e ‘ac’ operam na frequência de 5,8 GHz, embora o padrão ‘n’ possa operar também na frequência de 2,4 GHz, ou seja, suportar o padrão IEEE 802.11n não é garantia de que o equipamento trabalhe na frequência de 5,8 GHz. Os padrões IEEE 802.11 n e ac podem operar também na faixa de 5,4 GHz, que tem outros 23 canais disponíveis, mas não é todo equipamento que suporta essa faixa.

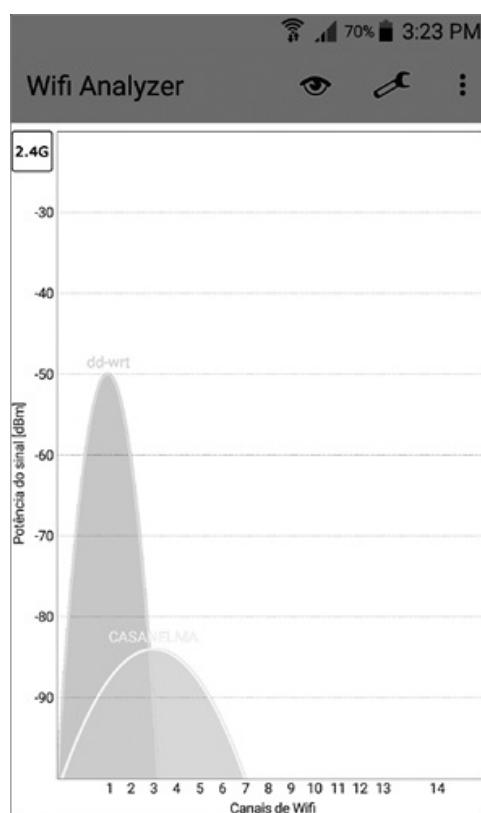



Figura 2.3 – Tela do aplicativo WiFi Analyzer indicando a sobreposição de canais.

2.3.2 Subcamada de acesso ao meio

A subcamada de acesso ao meio é responsável por decidir qual equipamento pode transmitir a cada momento. Ela é especialmente necessária em redes sem fio, visto que vários equipamentos estão sintonizados no mesmo canal ao mesmo tempo e só um pode ocupar o canal e transmitir num determinado momento.

Há vários tipos de protocolos e mecanismos para acesso ao meio. Em sistemas ponto a ponto, em que apenas dois equipamentos se comunicam, cada equipamento tem seu canal disponível para transmitir quando quiser, sem problemas com interferência. Nos sistemas ponto-multiponto, em que vários equipamentos podem receber as informações transmitidas, como é o caso da maioria das aplicações WiFi, deve haver regras para definir quem pode transmitir em cada momento.

As redes WiFi utilizam um protocolo de acesso ao meio do tipo CSMA/CA (Carrier sense multiple access with collision avoidance – Acesso múltiplo com verificação de portador e eliminação de colisão). É um mecanismo no qual todos os equipamentos que desejam transmitir ligam seus rádios e escutam o canal configurado antes de transmitir, para verificar se existe outro equipamento transmitindo. Se houver, aguarda a conclusão e liberação do canal. Quando o canal está liberado, o equipamento deve enviar, antes de transmitir, um pedido de transmissão a seu destinatário. Só após receber a confirmação do seu destinatário é que pode iniciar a transmissão. Isso é necessário porque seu destinatário pode se encontrar no alcance de um terceiro rádio que esteja transmitindo. A figura 2.4 apresenta essa situação. Se o AP estiver recebendo dados do computador A, ele não poderá responder à requisição de B. O computador B não consegue perceber que A está transmitindo, pois está fora do alcance de A.

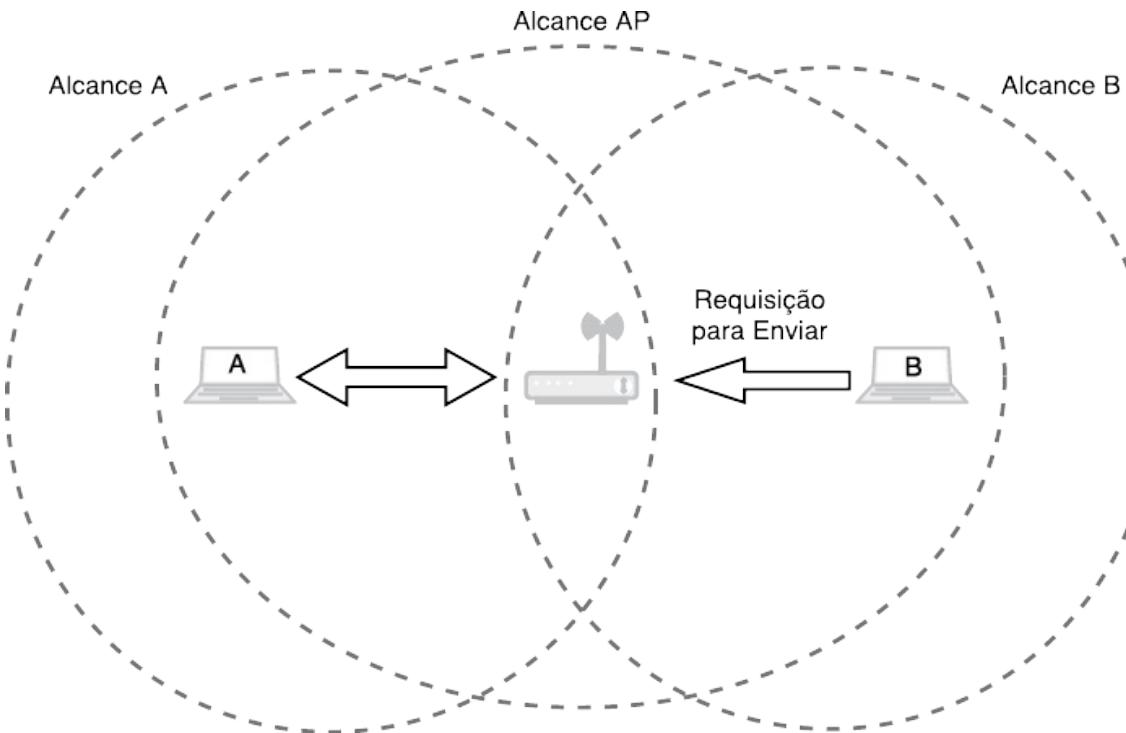



Figura 2.4 – Funcionamento do protocolo CSMA/CA.

O CSMA/CA é muito bom em situações nas quais poucos equipamentos demandam transmissão. Quando vários equipamentos têm muitas demandas de transmissão, os canais não conseguem ser devidamente reservados em virtude do grande número de pedidos de transmissão simultâneos. Na prática, poucos equipamentos com demandas altas ou algumas dezenas de equipamentos com demanda eventual conseguem congestionar um canal WiFi tornando seu uso inviável. Deve-se considerar ainda que a maioria dos roteadores WiFi, também conhecidos como AP (Access Point – Ponto de Acesso), de baixo custo tem um hardware muito limitado, que demora a responder, com poucos recursos para armazenar pacotes e tabelas até que possam ser transmitidos ou processados. Muitos roteadores simplesmente travam quando têm uma demanda alta.

É muito comum que projetos e propostas de redes WiFi fracassem por não considerar o dimensionamento correto dos equipamentos e canais. Não são raros os eventos e espaços públicos com anúncios de WiFi disponível, mas que, na prática, ninguém consegue acessar. Com um grande número de equipamentos solicitando permissão para transmitir, a maioria não consegue nem mesmo se registrar na rede.

Para um bom projeto de rede WiFi é preciso considerar uma série de

fatores:

- **Número e perfil de usuários** – Poucos usuários com perfil de alta demanda, como por exemplo fluxo de vídeo HD, podem congestionar um canal. Para alta demanda, deve-se considerar poucos usuários por canal. Para demanda eventual, como acessos em smartphone e aplicativos de mensagens, algumas dezenas de usuários podem ser suportados em um canal. Para demanda esporádica, como um dispositivo IoT que envie uma mensagem a cada período de tempo da ordem de segundos, algumas centenas de dispositivos podem ser alcançados em um mesmo canal.
- **Qualidade dos equipamentos** – Apesar de alguns APs baratos atenderem razoavelmente em uma pequena rede residencial, é preciso considerar equipamentos de melhor qualidade, com processadores mais rápidos, maior capacidade de memória e melhores rádios. Bons equipamentos podem ser encontrados a preços razoáveis. Vale a pena analisar as avaliações disponíveis em bons sites e revistas.
- **Distribuição dos canais pela área de interesse** – O dimensionamento das redes WiFi deve considerar a área na qual o serviço será oferecido. A área deve ser dividida em pequenas células. Cada célula deve ter seu número de usuários restrito, de acordo com o seu perfil. Logo, em áreas com grande concentração de usuários, as células devem ser reduzidas para que o número de usuários na célula não ultrapasse os limites para a qualidade de serviço desejado. O sinal de dois APs no mesmo canal não deve coincidir em nenhum ponto. Em qualquer ponto da área de oferta do serviço, apenas um AP deve operar cada canal. Assim, evita-se interferência entre dois APs diferentes.
- **Potência e antenas adequadas** – A potência de transmissão, acrescida do ganho agregado pela antena, deve determinar o alcance do sinal do AP. Assim, em áreas com grande concentração de usuários, um grande número de APs com potência reduzida deve ser a melhor configuração. Dessa forma, cada AP cobre uma pequena área com um número limitado de usuários. E as adjacências devem ser cobertas por outros APs em canais diferentes, evitando, assim, interferência entre eles.

A tabela 2.2 ilustra como seria uma boa divisão de canais entre 36 APs em

uma área quadrada. Para o melhor funcionamento, a potência dos APs deve ser configurada de forma que seu alcance não afete as regiões dos APs em diagonal, visto que utilizam o mesmo canal. Foram usados 12 APs em cada canal, considerando os três canais que não têm sobreposição (1, 6 e 11).

Tabela 2.2 – Distribuição ideal de canais em uma área dividida em 6 x 6 células

1	6	11	1	6	11
6	11	1	6	11	1
11	1	6	11	1	6
1	6	11	1	6	11
6	11	1	6	11	1
11	1	6	11	1	6

O projeto de redes WiFi não é tão trivial quanto se espera ao comprar um roteador em promoção. Os exemplos de projetos frustrados são mais comuns que os projetos bem-sucedidos, especialmente em espaços mais densos, como prédios de apartamentos, escolas, shoppings ou empresas. O problema se torna maior porque os equipamentos são sempre configurados com a potência mais alta possível, além de haver upgrades de antenas na expectativa de melhorar a rede. E só piora, visto que a interferência com os equipamentos vizinhos vai só aumentar. É nesse cenário que a IoT será inserida. Contudo, pode se tornar um fracasso se não houver uma organização eficiente das redes WiFi e de seus usuários.

2.4 Modos de comunicação WiFi

2.4.1 Infraestrutura fixa

O modo-padrão de comunicação WiFi utilizado na maioria das aplicações é chamado de "infraestrutura fixa" devido à presença de um ponto de acesso (AP) que tem por finalidade concentrar a comunicação de rede WiFi. Nesse modo, toda comunicação deve passar pelo AP, que identifica seu destinatário e repassa a comunicação.

O modo de infraestrutura fixa tem como maior vantagem a presença de um

elemento que pode definir parâmetros gerais, como segurança, canal utilizado, versão do WiFi, distribuição de endereços e NAT. Uma possível desvantagem pode ser vista na comunicação direta entre dois equipamentos na mesma rede. Como a comunicação deve, obrigatoriamente, passar pelo AP, o tráfego na rede WiFi é dobrado. Por exemplo, caso um equipamento A se comunique com B, todo pacote deve seguir no sentido A → AP e depois AP → B, dividindo a capacidade do canal de comunicação por dois. Na prática, no entanto, isso não é percebido, pois a maioria das comunicações tem o AP como origem ou destino.

2.4.2 Ad hoc

O modo de comunicação *ad hoc* exclui a necessidade de um ponto de acesso. Os elementos de rede podem se comunicar diretamente desde que sejam configurados com os mesmos padrões de rede, que incluem, normalmente, o identificador de rede WiFi (SSID) e o endereçamento IP. Uma rede *ad hoc* pode conter apenas elementos que se comuniquem de forma direta desde que estejam no alcance do rádio, ou pode incluir algoritmos de roteamento entre os elementos de rede, que podem atuar como roteadores, encaminhando os pacotes ao seu destino.

Diversos algoritmos de roteamento em redes *ad hoc* estão propostos na literatura para encaminhar pacotes em redes *ad hoc* entre nós que não estão em alcance direto, usando os demais elementos como roteadores. Entre os principais trabalhos estão o Dynamic source routing in ad-hoc network (DSR) [Sameh, 1996] e On-Demand Distance Vector Routing (AODV) [Johnson & Maltz, 1996]. Em ambos os casos, o caminho a ser percorrido entre os pacotes enviados pela rede será determinado de forma automática, através de buscas pela rede, visando localizar a forma mais rápida e que use menos recursos da rede.

2.4.3 WiFi Direct

WiFi Direct é uma tecnologia similar ao modo *ad hoc*, visto que tem a primeira finalidade de dispensar a necessidade do roteador ou AP WiFi na rede. Com algumas diferenças: não há previsão de roteamento entre os nós, ou seja, eles precisam estar no raio de alcance para comunicação

direta; e a configuração é automática, utilizando alguns protocolos de descoberta direta de dispositivos. Resumindo, o WiFi Direct não requer que o usuário insira as configurações da rede WiFi e TCP/IP. Ele se configura automaticamente.

Diversos dispositivos já incorporaram o modo de comunicação WiFi Direct, entre eles, Smart TVs, smartphones, tablets e sistemas operacionais como Windows, Mac ou Linux. Isso permite que eles comuniquem diretamente, sem a necessidade de roteador, tampouco de configuração prévia. Basta habilitar o modo WiFi Direct e pesquisar por outros dispositivos que tenham essa função na rede. Uma vez conectados, os dispositivos podem compartilhar arquivos e demais fluxos de comunicação.

WiFi Direct é similar ao protocolo Bluetooth, executando basicamente a mesma função, porém a velocidades bem maiores. Na prática, foi criado para atender às mesmas aplicações do Bluetooth, com maiores velocidades.

No cenário de IoT, WiFi Direct pode ser muito utilizado, especialmente para descobrir os dispositivos e serviços disponíveis e acessá-los de dispositivos como smartphones ou Smart TVs. A figura 2.5 mostra a tela de configuração do sistema Android para smartphone localizando dispositivos com o recurso WiFi Direct no alcance do smartphone.

Figura 2.5 – Tela de configuração do WiFi Direct do Android.

2.4.4 Redes Mesh

É comum, no cenário de IoT, que os diversos dispositivos fiquem dispersos por uma grande área, de forma a tornar difícil deixar todos dentro do alcance da rede sem fio. Uma alternativa para ampliar o alcance do rádio e incluir nós que estejam fora do alcance dos APs é usar o modo Mesh, no qual todos os dispositivos passam a atuar também como roteadores, encaminhando mensagens de outros dispositivos para um AP que esteja a seu alcance ou, ainda, a outro dispositivo que esteja mais próximo de um AP. A figura 2.6 apresenta a possibilidade de usar o módulo ESP8266 na configuração de rede Mesh.

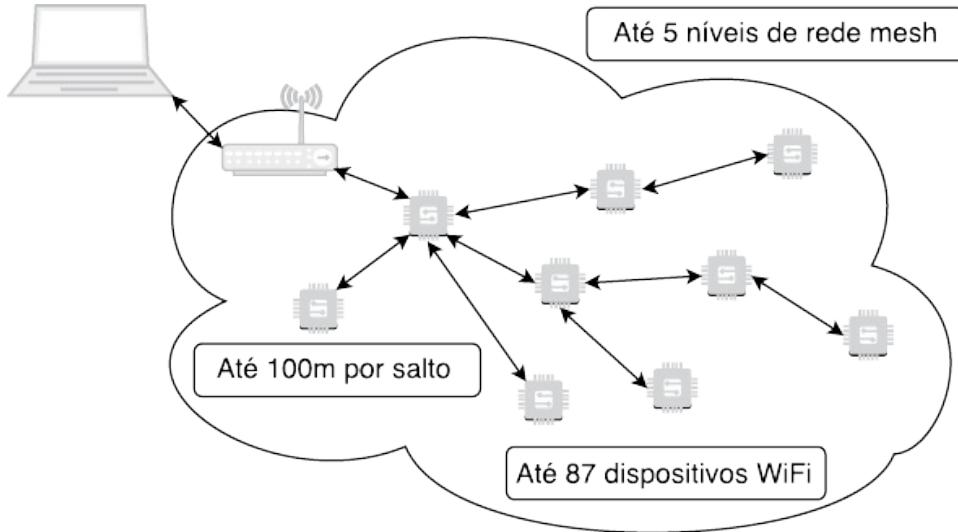



Figura 2.6 – Características da rede Mesh do ESP8266.

Nessa configuração, uma informação pode ser transportada em vários saltos sem fio, passando por vários dispositivos, até chegar a um AP que a encaminhe para seu destino na internet. Apresenta como vantagens a possibilidade de incluir dispositivos IoT em áreas onde não há rede disponível.

A configuração em rede Mesh não é exclusividade das redes WiFi. Sua proposta originou-se nas redes *ad hoc*. As redes *ad hoc* apresentam, ainda, pouca aplicação prática. As redes Mesh, por sua vez, têm sido amplamente usadas em soluções IoT.

2.5 RFID

A tecnologia conhecida como RFID (Radio Frequency Identifier – Identificador por Radiofrequência) é bem antiga, data de meados de 1940. Foi usada, inicialmente, nos aviões de guerra para facilitar sua identificação em combate. Ainda é usada na aviação até os dias atuais e serve para informar sobre a presença de aviões.

A essência da tecnologia é basicamente transmitir um código identificador por um canal de radiofrequência que pode ser associado a um objeto ou uma "coisa". Um equipamento receptor associa o código transmitido com o objeto. E pode ter inúmeras aplicações no cotidiano, desde a substituição dos códigos de barra nos produtos do supermercado, passando pela identificação e cobrança automática de carros nos pedágios e

estacionamentos, bem como o controle de acesso e cobrança em catracas de ônibus, clubes, bares etc.

As aplicações de RFID vão determinar o tipo a ser usado, que pode pertencer a uma de duas classes: ativas ou passivas. A tecnologia RFID ativa inclui uma fonte de alimentação, seja por baterias ou por meio de uma fonte constante. A disponibilidade de energia possibilita que o sinal de radiofrequência transmitido tenha uma potência maior e, assim, alcance maior distância. Não há limites para essa distância. No caso dos aviões, por exemplo, pode alcançar quilômetros, ou uma dezena de metros, para a identificação de carros em pedágios e estacionamentos.

A tecnologia RFID passiva não inclui fonte de alimentação. A alimentação é recebida também sem fio. O chip RFID, também chamado de etiqueta ou *tag*, consegue receber energia suficiente para se alimentar e transmitir, em resposta, sua identificação. Assim, pode ser bem leve, compacto e barato. Uma tag RFID pode ser tão pequena quanto um adesivo com um código de barras e custar centavos.

As aplicações RFID se tornam ainda mais interessantes porque os leitores RFID podem ser capazes de ler várias tags em pouco tempo, favorecendo aplicações por meio das quais muitas "coisas" podem ser identificadas de forma quase imediata. Assim, um carrinho de supermercado cheio de produtos com tags RFID seria faturado de forma quase imediata, sem que fosse necessário retirar os produtos do carrinho. Da mesma forma, um leitor em um armário ou uma geladeira faria um balanço de todo o estoque em um tempo ínfimo. Isso pode render várias aplicações de IoT.

As tags e os leitores RFID podem trabalhar em três conjuntos de frequências: baixa frequência (LF), de 125 kHz, alta frequência (HF), 13,56 MHz, e ultra-alta frequência (UHF), na faixa de 860 a 960 MHz. Os leitores e tags nas frequências mais baixas são mais comuns e acessíveis. A frequência de 13,56 MHz tem, ainda, a vantagem de ser compatível com o padrão de comunicação NFC.

O padrão NFC (Near Field Communication – comunicação em campo próximo) foi projetado para comunicação entre dispositivos a distâncias muito curtas, da ordem de centímetros. Está disponível em vários tipos de smartphones e equipamentos portáteis, e suas aplicações são mais

diversificadas que a tecnologia RFID, visto que pode ser usado também para transferir arquivos ou manter a comunicação de quaisquer outras aplicações. Seu diferencial é oferecer um serviço de comunicação a uma distância muito curta, o que exigiria a intervenção e a intenção direta do usuário, elevando seu grau de segurança. As aplicações típicas são para transações comerciais, como transferência de recursos, possivelmente substituindo cartões de crédito e outras formas de pagamento.

O compartilhamento da frequência de 13,56 MHz entre RFID e NFC é intencional. Há vários dispositivos leitores que suportam RFID e NFC como alternativa às tags RFID, especialmente em sistemas de controle de acesso. Em vez de levar várias tags RFID, um usuário poderia substituí-las pelo seu smartphone, já antecipando que a utilização de RFID para controle de acesso nos diversos ambientes e sistemas pode exigir que um usuário tenha mais tags RFID que chaves em seu chaveiro.

As tecnologias RFID e NFC são fundamentais para IoT. São usadas em aplicações de controle de acesso, logística, controle de estoque, transferência de recursos, comércio eletrônico e físico, entre outros tipos que se possam inventar agregando as tecnologias RFID e NFC às demais tecnologias embarcadas e com comunicação sem fio previstas em IoT.

2.6 Protocolos específicos para IoT

Os protocolos de comunicação apresentados até aqui não foram desenvolvidos para Internet das Coisas. Soluções comerciais, como WiFi, existem há mais de dez anos, quando não se conheciam termos como Internet das Coisas, tampouco Redes de Sensores. No entanto, como são soluções comerciais e estão disponíveis a baixo custo, se tornaram imediatamente viáveis para a utilização em IoT. Contudo isso não significa que sejam os protocolos ideais para o ambiente.

Alguns protocolos foram propostos exclusivamente para o ambiente de IoT, ou ainda para Redes de Sensores Sem Fio, que têm um perfil parecido com IoT, como baixa demanda de largura de banda, presença de grande número de dispositivos na rede, alcance limitado a poucos metros e necessidade de economia de energia. Percebe-se que esses objetivos são muito diferentes dos objetivos previstos inicialmente em redes WiFi, que

foram projetadas para acesso de dispositivos móveis em redes locais.

Os protocolos da família IEEE 802.15 foram projetados para as redes conhecidas como WPAN (Wireless Personal Area Network – redes sem fio que alcançam poucos metros de distância). O protocolo Bluetooth está incluído nesse padrão e pode ser usado em muitas outras aplicações, além de fazer conexão entre um smartphone e um dispositivo IoT. Também fazem parte desse padrão os protocolos ZigBee, 6LoWPAN, WirelessHart e ISA100.11a.

O protocolo Bluetooth, definido no padrão IEEE 802.15.1, foi o primeiro protocolo WPAN a se tornar comercial e se popularizar. Inicialmente, sua aplicação para telefones celulares, que possibilita a troca de arquivos diretamente entre os dispositivos, se tornou bastante útil e se difundiu de forma a se tornar requisito obrigatório em todos os telefones numa época em que conexão WiFi ou 3G ainda não era realidade. Outra aplicação que se popularizou para o Bluetooth foi a ligação de periféricos como fones de ouvido, sistemas de som, teclados e mouse, tanto a celulares quanto a computadores portáteis.

Uma variação do Bluetooth é conhecida como BLE (Bluetooth Low Energy) e tem como característica principal o consumo reduzido de bateria, o que o habilita para aplicações que exigem um tempo de vida elevado para a bateria. Sistemas de localização e rastreamento são exemplos de aplicação de BLE, aprimorando recursos já disponíveis pela tecnologia RFID, mas com maiores restrições de alcance.

O padrão **ZigBee** utiliza os protocolos definidos no padrão 802.15.4 na camada de enlace e na camada física. Suas características incluem baixo consumo de energia, baixa taxa de transmissão e baixo custo de implementação. O alcance é limitado a, no máximo, cerca de 100 metros. Para atingir distâncias maiores, o ZigBee conta com o repasse das informações pelos nós da rede, em múltiplos saltos, até alcançar o seu destino. Algumas soluções ZigBee comerciais estão disponíveis no mercado, sendo a mais comum conhecida como **XBee**.

Outra alternativa para Internet das Coisas é o **6LoWPAN,** que significa IPv6 over Low power Wireless Personal Area Networks. Sua proposta é encapsular o protocolo IPv6 para redes sem fio de curto alcance, usando o

padrão IEEE 802.15.4 nas camadas de enlace e física. Para isso, os cabeçalhos IPv6 devem ser fragmentados, compactados e reagrupados. Não é uma questão fácil, pois os cabeçalhos IPv6 são extensos. Contudo ele possibilita que cada dispositivo IoT tenha um endereço IPv6 único. Na prática, no entanto, os dispositivos 6LoWPAN ainda não se tornaram comerciais e viáveis para IoT, da mesma forma que IPv6 também ainda não se popularizou nem para a internet convencional.

Uma tecnologia que se tornou comercial para IoT na indústria é conhecida como **WirelessHart** e também usa o padrão IEEE 802.15.4 nas camadas inferiores. Essa tecnologia é baseada no protocolo HART (Highway Addressable Remote Transducer Protocol), uma implementação de *FieldBus*, um padrão digital de redes industriais. Esses protocolos são amplamente usados na indústria e têm interoperabilidade com os equipamentos de automação, como CLP (Controlador Lógico Programável), e os sistemas supervisórios, usados para criar as interfaces para os operadores de plantas industriais automatizadas.

Em 2015, outra proposta de protocolo foi desenvolvida, para atender a redes IoT de longa distância. Conhecida como **LoRaWAN** (Long Range Wide Area Network), essa proposta está focada em acessos a dispositivos a até 15 quilômetros de distância, usando uma estação de rádio de longo alcance. A proposta parece atender a requisitos de aplicações de tarifação remota de serviços residenciais, como água, luz e gás encanado. Esse tipo de acesso demanda uma banda mínima de comunicação, mas é necessário um longo alcance, visando atingir todas as residências dentro de uma região com o menor investimento possível. Para atingir distâncias maiores, a proposta conta com frequências menores, entre 433 e 915 MHz, frequências que também são abertas, mas têm baixa largura de banda e poucos canais.

As opções de redes para Internet das Coisas ainda não estão bem definidas. No momento, WiFi e Bluetooth têm sido usados em várias aplicações e estão se consolidando, ganhando mercado e disponibilizando produtos a baixo custo. Começam a surgir aplicações e cobertura para outros tipos de redes, como a LoRaWAN, já com cobertura em grandes áreas em países desenvolvidos. Por cobrir áreas maiores, esse tipo de rede pode ter grande utilidade no futuro, em aplicações de telemetria, medição de consumos de

água, luz e gás residenciais, trânsito e agropecuária, entre outras funções.

CAPÍTULO 3

Arquitetura de sistemas embarcados

Este capítulo é dedicado a apresentar conceitos de microcontroladores e placas usados em sistemas embarcados, com enfoque nas placas que serão abordadas no livro: ESP8266, Arduino e Raspberry PI.

3.1 Microcontroladores

Desde meados de 1970, quando a Intel iniciou sua linha de microprocessadores de baixo custo – 4004, 4040, 8008, 8080 e 8051 – e seus concorrentes, em especial a Texas Instruments com seu microcontrolador TMS 1000, que os sistemas embarcados começaram a se desenvolver e ocupar tarefas que antes eram feitas de forma manual. Alguns desses componentes começaram a se destacar na automatização dos dispositivos e sistemas por agregar mais funcionalidades no próprio componente, recebendo o nome de SoC (System-on-Chip). Dispositivos com essas características começaram a ser chamados de microcontroladores devido ao seu uso na função de controle e automação.

Os microcontroladores têm interface de entrada e saída com dispositivos elétricos como botoeiras e relés, ao contrário dos microprocessadores de uso geral, normalmente dedicados a projetos de computadores com periféricos de entrada e saída como teclado, mouse e monitor. As aplicações de microcontroladores, inicialmente incipientes e voltadas para a indústria, começaram a se propagar nas diversas áreas de atividade humana.

As funcionalidades agregadas aos microcontroladores inicialmente se resumiam a interfaces de entrada e saída (I/O) e foram agregando, a cada nova versão ou produto, memória RAM, memória EPROM para programas e dados e circuito de oscilador (clock), interfaces de comunicação (serial, USB) e, mais recentemente, interfaces de rede, Ethernet, WiFi e Bluetooth.

Um microcontrolador com todas essas funcionalidades integradas é capaz de atender a um número de aplicações sem precedentes. A única coisa de que ele precisa é energia, e, ainda assim, muito pouca energia, visto que vários microcontroladores atuais são classificados como de baixíssimo consumo de energia. Ou seja, uma bateria poderia mantê-los em funcionamento por um bom tempo, sem necessidade de carga ou substituição.

A figura 3.1 apresenta o diagrama funcional do microcontrolador ESP8266, da Espressif. Como se pode perceber, há um conjunto completo de componentes para processamento (CPU + memória), interfaces de entrada e saída para diversos tipos de protocolos e aplicações, e comunicação.

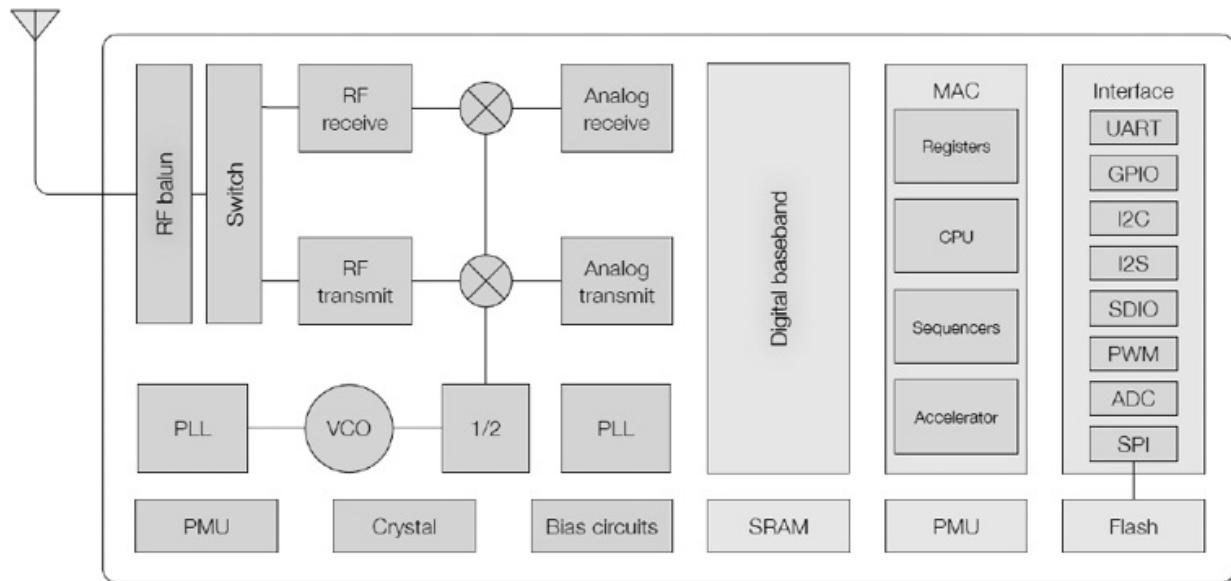



Figura 3.1 – Arquitetura interna do microcontrolador ESP8266.

A junção de todas essas funcionalidades em um único componente torna a sua integração muito barata e acessível, o que justifica sua implantação em dispositivos de baixo custo, como uma torradeira, um forno, uma fechadura ou um alarme. Por outro lado, o componente poderia se tornar caro, mas a indústria de semicondutores e sua escala de produção sempre conseguem derrubar os preços de componentes que têm alta demanda. A alta escala de produção derruba os preços ao dividir os custos de projeto e implantação do processo de fabricação. Em comparação com os microprocessadores voltados para o mercado específico de computação pessoal, os microcontroladores têm um poder de processamento muito

inferior, em contrapartida, ao número de recursos embutidos no componente.

A história de microcontroladores de destaque inclui o **8085** e o **Z80**, nas décadas de 70 e 80, o **Microchip PIC,** na década de 90 e no início do século XXI, e o **Arduino**, na última década, que tem como maior legado seu ambiente de desenvolvimento, com diversas bibliotecas em linguagem C, incluindo suporte a diversos tipos de módulos de sensores, atuadores e comunicação.

3.2 Interfaces de entrada e saída

A grande vantagem dos microcontroladores é integrar, no mesmo componente, as diversas interfaces de entrada e saída (I/O), além de outras funcionalidades, como o circuito de oscilador e memória. As interfaces de entrada e de saída são cruciais para definir se um determinado microcontrolador atende a um projeto de integração de sistemas embarcados.

As interfaces de entrada e de saída mais comuns são conhecidas como **GPIO** (General Purpose Input Output). Podem ser configuradas como entradas ou saídas, por software, e representam interfaces digitais simples, que normalmente assumem o valor 0 ou 1, representando nível de tensão baixo ou alto, respectivamente, na maioria dos casos. Há casos de lógica inversa, nos quais a tensão baixa representa 1 e a tensão alta representa 0. Também há sistemas que conseguem detectar entradas em aberto, com impedância alta, como uma terceira alternativa para valor de entrada.

Uma aplicação típica de uma GPIO como entrada é ligar uma botoeira. Nesse caso, um resistor é necessário para manter a entrada em baixa tensão (terra), enquanto a botoeira não é acionada. Seu acionamento leva a tensão de entrada para nível alto, que é recebido na entrada do microcontrolador. Como saída, as GPIO podem ser usadas para acionar dispositivos, acender leds ou acionar outros componentes. Vale lembrar que, em geral, a corrente de saída é baixa, exigindo a utilização de um relé, responsável, então, pelo acionamento.

Interfaces de **entrada analógica** também são comuns em microcontroladores, embora não estejam presentes em todos os modelos.

São entradas que podem receber uma tensão de entrada variável, que pode ser recebida pelo software como um valor proporcional à tensão. Um caso típico que serve como exemplo é a entrada analógica do Arduino, que pode receber uma tensão entre 0 e 5 volts, que pode ser lida por software, resultando num valor inteiro de 10 bits, correspondendo a um número entre 0 e 1.023, diretamente proporcional à tensão de entrada. Entradas analógicas são úteis para receber informações de sensores analógicos, como sensores de temperatura, pressão, umidade etc.

A interface de **saída PWM**, presente na maioria dos microcontroladores, é uma saída digital, porém, de frequência e largura de pulso variável, que, na prática, se comporta com uma saída analógica. A figura 3.2 mostra um exemplo de sinal PWM. A saída PWM é, normalmente, ligada a um dispositivo de controle, como um relé de estado sólido ou um driver de motor. Este pode ser usado para controlar a potência, a tensão ou a velocidade de um dispositivo analógico, como um motor ou um aquecedor.

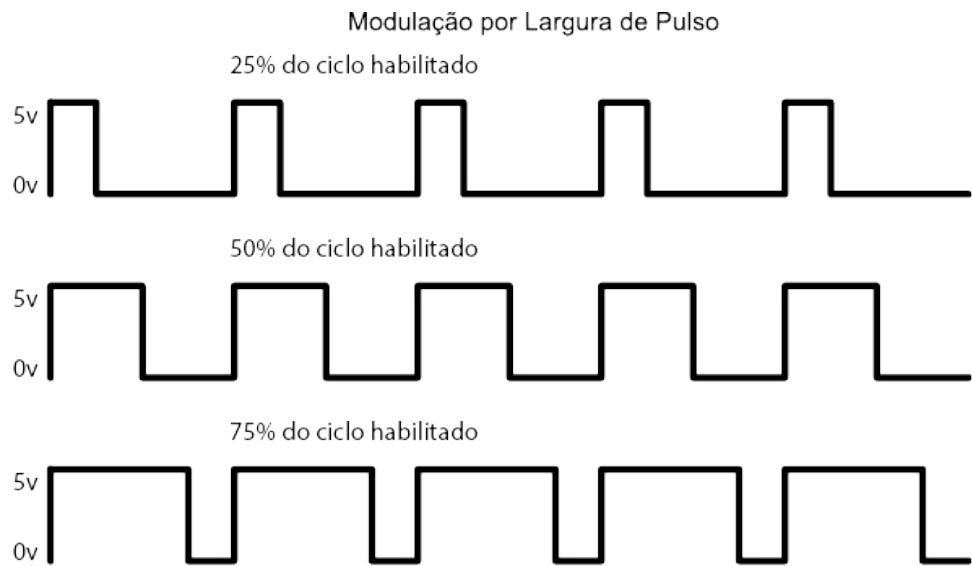



Figura 3.2 – Exemplos de saída PWM.

Além das interfaces de entrada e saída usadas para receber e enviar informações de outros componentes sem processamento, os microcontroladores também podem ter interfaces de comunicação, que são capazes de trocar informações com outros dispositivos microprocessados, que possam se comunicar por meio de um protocolo de comunicação previamente estabelecido.

3.3 Interfaces de comunicação

Várias interfaces de comunicação podem estar disponíveis nos microcontroladores, desde interfaces seriais até interfaces de rede sem fio. As interfaces seriais são mais comuns e antigas, desde interfaces RS-232 até as interfaces USB. Já as interfaces de rede, podem possibilitar a comunicação entre vários dispositivos em uma rede, seja com fio, normalmente no padrão Ethernet, ou sem fio, como WiFi ou Bluetooth.

As **interfaces seriais** servem para ligar dois dispositivos de forma direta, em uma comunicação ponto a ponto. São comuns para interligar dois sistemas microprocessados que precisam trocar informações ou para ligar periféricos como teclados, telas e redes de sensores inteligentes.

As interfaces seriais podem ser divididas em interfaces síncronas ou assíncronas.

As interfaces síncronas devem compartilhar o relógio para que possam fazer a leitura e a escrita ao mesmo tempo. Para isso, normalmente têm um pino adicional para compartilhar o relógio. São exemplos de seriais síncronas as interfaces SPI e I2C.

As interfaces seriais assíncronas não compartilham um sinal de relógio. Para identificar o início e o final de cada transmissão, há um sinal de partida (inicialização) e outro de parada (finalização). As interfaces RS232, RS485 e USB são assíncronas.

As **interfaces de rede** podem interligar vários dispositivos em uma rede, possibilitando a comunicação entre todos eles, especificando o endereço do dispositivo a comunicar, ou até para todos de forma simultânea, indicando o tipo de comunicação conhecida como *broadcast*. A interface de rede mais comum é o padrão **Ethernet**, definido no padrão IEEE 802.3, que prevê as duas camadas inferiores da pilha de protocolos TCP/IP. Não é exigido, no entanto, que toda a pilha TCP/IP seja usada. Existem aplicações em que a comunicação é feita diretamente ao protocolo Ethernet, usando o endereçamento previsto nesse protocolo.

As interfaces de rede **WiFi** e **Bluetooth** tornaram-se mais comuns nos últimos anos devido graças à sua popularização e ao desenvolvimento dos circuitos eletrônicos, sendo capazes de produzir microcontroladores com

essas interfaces já embutidas. Os microcontroladores que não têm essas interfaces embutidas podem acoplar módulos externos que implementam esse tipo de comunicação. Normalmente, a comunicação entre esses módulos e o microcontrolador se dá por uma interface serial.

3.4 Arduino

Para começar a falar do Arduino é preciso esclarecer que não se trata de um microcontrolador, mas, sim, de uma plataforma de desenvolvimento de sistemas embarcados de baixo custo aberta e livre. Assim, não está vinculado a nenhum fabricante específico, embora a maioria dos módulos disponíveis utilize microcontroladores da Atmel.

As referências a Arduino consideram, normalmente, uma placa integrada com um microcontrolador e suas interfaces de entrada e saída, alimentação e comunicação. O módulo mais comum é o Arduino Uno, basicamente, um kit de desenvolvimento para iniciantes. Há diversos outros módulos Arduino, entre os quais, o Arduino Nano, módulo mais voltado para a produção de sistemas embarcados, além de diversos outros módulos compostos de sensores, atuadores e comunicação.

O Arduino teve a primazia de tornar o desenvolvimento de sistemas embarcados acessível a toda espécie de curiosos, entusiastas e estudantes de elétrica, eletrônica e computação. Antes do Arduino, desenvolver um sistema embarcado exigia a compra de um kit de desenvolvimento nem sempre barato, com recursos para programação e gravação dos microcontroladores, além da necessidade de construir uma placa para acomodar os componentes. As placas Arduino vêm prontas para utilização e integração e têm baixo custo; o ambiente de desenvolvimento torna a tarefa mais fácil.

A figura 3.3 mostra a interface da IDE (Integrated Development Enviroment) Arduino, que está disponível no site principal do Arduino: *https://www.arduino.cc/*. A linguagem de programação é C. Há milhares de exemplos e tutoriais disponíveis na internet, iniciando pelo site principal do Arduino.

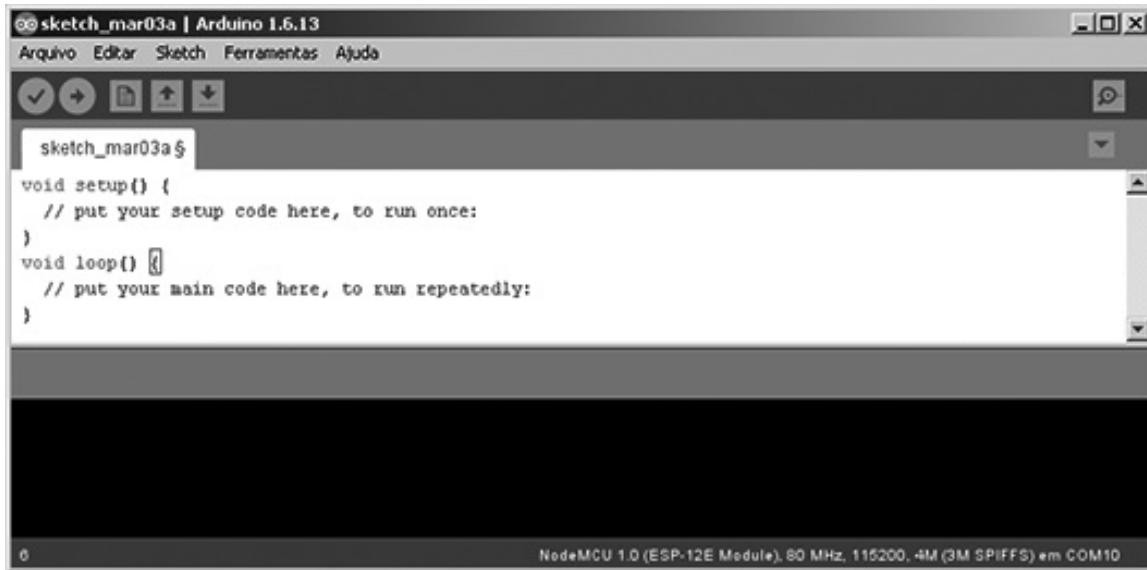



Figura 3.3 – Interface da IDE Arduino.

3.4.1 Módulos periféricos Arduino

A popularização do Arduino trouxe algumas vantagens importantes para o seu desenvolvimento. Vários módulos, conhecidos como *shields*, começaram a ser produzidos e consumidos em larga escala, tornando-se também acessíveis e muito úteis na integração de sistemas embarcados. Há vários módulos disponíveis, incluindo sensores, atuadores e módulos de comunicação.

Entre os módulos sensores, é possível destacar o módulo **sensor de corrente elétrica**, módulo de **sonar**, módulo **detector de movimento**, módulo **sensor de umidade do solo**, módulo **sensor de temperatura e umidade do ar**, módulo **sensor de gases inflamáveis**, módulo **sensor de luminosidade**, módulo **sensor infravermelho**, entre dezenas de outros módulos extremamente úteis em aplicações de Internet das Coisas.

Entre os módulos atuadores, os mais comuns são os módulos de **relé**, que podem aparecer em diversas configurações de número de canais, de um a oito canais. Esses módulos são ótimos para controlar diversos tipos de dispositivos com função de liga/desliga, como lâmpadas, motores e aparelhos eletrônicos. Também são comuns os módulos de controle de servomotores, usados especialmente em robótica.

Os módulos de comunicação incluem módulos **WiFi, Ethernet** e **Bluetooth**. Esses módulos servem para interligar o Arduino a outros

dispositivos em aplicações de Internet das Coisas. Os módulos **WiFi** e **Ethernet** são usados para ligar o Arduino à internet. Para isso estão disponíveis diversas bibliotecas com as pilhas de protocolo TCP/IP e WiFi, que possibilitam a interligação nos diversos modos de WiFi, bem como algumas aplicações TCP/IP, como a web. Há bibliotecas para que o Arduino funcione como cliente web, enviando solicitações e acessando dados na web, ou como servidor, disponibilizando acesso às suas informações por meio do protocolo HTTP.

O módulo **Bluetooth** é particularmente interessante para comunicação direta com aplicativos em smartphones, que normalmente suportam esse protocolo. Isso permite controlar e acessar informações de diversos dispositivos, desde que estejam próximos, visto que o alcance do Bluetooth é da ordem de poucos metros. O Bluetooth também pode ser usado para acessar periféricos de entrada e saída, como teclados, mouse, microfones e alto-falantes. Nesse caso, no entanto, está limitado àqueles periféricos que são suportados pelas bibliotecas do Arduino.

3.5 Raspberry Pi e placas Linux

O poder de processamento de microcontroladores é, normalmente, bem limitado. A maioria dos microcontroladores tem palavras de 8 bits e executa com poucos megahertz de frequência. Isso limita muito o poder de processamento das aplicações nos sistemas microcontrolados. Essa limitação abre espaço para placas com microprocessadores mais velozes, como os processadores da família x86 e ARM. Entre estes, o destaque é o módulo conhecido como Raspberry Pi.

O projeto Raspberry Pi é também um projeto de hardware aberto que utiliza um processador da família ARM. Atualmente na versão 3, tem um processador de 1,2 GHz de 64 bits, incluindo, ainda, WiFi, Bluetooth, um gigabyte de memória RAM, quatro portas USB, 40 pinos de entrada e saída, saída HDMI e diversas outras interfaces, como câmera e memória externa. Tem, ainda, um processador gráfico capaz de codificar vídeo H.264 full HD em tempo real. Como todo esse poder de processamento e interface, suas aplicações são ilimitadas.

Inicialmente, o Raspberry Pi foi proposto como uma solução de

computador popular, executando uma versão do sistema operacional Linux, com interface gráfica e pacotes de aplicativos de código aberto. Seu sucesso, no entanto, abriu espaço para várias outras aplicações, especialmente em sistemas embarcados. Suas interfaces de entrada e saída, bem como seu tamanho reduzido e as demais interfaces, facilitam vários tipos de aplicações.

O sistema operacional Linux possibilita, ainda, a integração de vários tipos de periféricos cujos controladores estão disponíveis para esse sistema, incluindo câmeras, teclados, mouses, impressoras, monitores e telas, scanners, leitores biométricos, entre outros. Logo, as aplicações podem ser incrementadas com diversos tipos de dispositivos.

A disponibilidade de software para o sistema Linux que pode ser usado no Raspberry Pi é uma grande vantagem dessa placa. Ambientes de programação nas mais diversas linguagens, incluindo interfaces gráficas e APIs para integração de uma infinidade de dispositivos, além de servidores de bancos de dados, servidores e protocolos de rede para as mais diversas aplicações, estão disponíveis para o sistema Linux e podem ser executados na Raspberry Pi. Várias aplicações de sistemas embarcados podem demandar esses recursos.

Algumas aplicações de Internet das Coisas podem contar com uma configuração de rede heterogênea, incluindo um Raspberry Pi como elemento central e vários outros dispositivos menores e mais baratos executando funções distribuídas na rede. Assim, aproveitam-se todos os recursos da Raspberry Pi sem aumentar demais o custo da rede.

Outras placas similares ao Raspberry Pi também estão disponíveis, entre as quais, Banana Pi, BeagleBone, Cubieboard e Pandaboard. Todas usam processadores ARM e suportam o sistema Linux, sendo que algumas também suportam o sistema Android, além de apresentar algumas interfaces adicionais. A BeagleBone, por exemplo, é base do projeto Erlebrain, de controlador para drones de alto desempenho.

3.6 ESP8266

O microcontrolador ESP8266, produzido pela fabricante chinesa Espressif,

é um microcontrolador de 32 bits que inclui um núcleo microprocessado Tensilica L106, que funciona na frequência-padrão de 80 MHz, podendo chegar a 160 MHz. O processamento da pilha de protocolos WiFi usa 20% da capacidade de processamento desse processador. Com isso, 80% dessa capacidade pode ser utilizada em aplicações do usuário. A memória disponível para os dados dos programas tem cerca de 50 kB, já descontado o espaço necessário para o padrão WiFi. A memória disponível para o programa principal é de 4 MB, em área acessível à atualização em funcionamento, também conhecida como OTA (Over-The-Air); além de mais 512 kB que não contam com esse recurso e só podem ser atualizados via cabo, em procedimento de atualização.

A tensão nominal de funcionamento é de 3,3 volts. O consumo de energia conta com 20 µA no modo *sleep* e cerca de 50 mA conectado a um AP WiFi em modo de recepção. Na transmissão em potência máxima de 17 dBm, a corrente de consumo pode chegar a 170 mA. Considerando uma bateria recarregável de 3,7 volts e 1.000 mAh de capacidade, pode manter o funcionamento do ESP8266 por cerca de vinte horas ininterruptas se não mantiver uma taxa alta de transmissão. É possível, ainda, considerar aplicações nas quais o microcontrolador entra no modo *sleep* sempre que não há demanda. Isso pode reduzir o tempo de funcionamento para uma pequena parcela do tempo, aumentando consideravelmente o tempo de vida de uma bateria alimentando esse microcontrolador.

Em relação às interfaces de entrada e saída, o ESP8266 tem 17 interfaces GPIO que podem ser configuradas como entradas ou saídas digitais. Tem também quatro interfaces de saída PWM e uma entrada analógica com 10 bits de precisão, por uma tensão de 3,3 volts, que é a tensão de funcionamento do ESP8266.

Como interfaces de comunicação, há interfaces seriais síncronas, SPI, I2C e I2S; e assíncronas USART, que podem ser usadas para interligar uma interface RS-232. E, claro, não podemos nos esquecer da interface WiFi, que pode atuar no modo AP, como cliente, no modo *ad hoc* ou WiFi Direct.

O ESP8266 tem, ainda, Atualização OTA (Over The Air), que possibilita que seu firmware seja atualizado pela rede WiFi, em vez de ser atualizado

exclusivamente pela porta USB/Serial. Essa característica é muito importante na atualização de software para correção de erros, já depois de implantado. É possível que os dispositivos sejam atualizados periodicamente, assim como ocorre com os sistemas operacionais modernos.

O microcontrolador ESP8266 foi escolhido como base para as aplicações deste livro. Entre as diversas razões para a sua escolha estão:

- **Preço** – um módulo com placa de circuito impresso e antena na placa pode ser comprado em sites chineses por cerca de 3 dólares, pronto para uso, o que o torna potencial para aplicações de baixo custo, como se espera de Internet das Coisas, facilitando sua integração a todo tipo de aplicação.
- **Ambientes de desenvolvimento** – três ambientes de desenvolvimento estão disponíveis para o ESP8266: i) um ambiente baseado na linguagem Lua, interpretado e de fácil integração; ii) o ambiente de desenvolvimento do Arduino, incluindo todas as suas bibliotecas e interfaces, herdando, assim, toda a base de conhecimento e aplicações já desenvolvidas para o Arduino; e iii) um ambiente de desenvolvimento baseado no sistema operacional de tempo real RTOS, capaz de criar aplicações profissionais, explorando todo o potencial do microcontrolador e suas interfaces.
- **Disponibilidade de módulos periféricos** – todos os módulos propostos e popularizados para o Arduino são compatíveis com o ESP8266. Isso implica em uma grande disponibilidade de placas, desde sensores a atuadores e ferramentas de comunicação diversas. A facilidade de integrar essas placas, especialmente no próprio ambiente de desenvolvimento do Arduino, propicia o desenvolvimento de aplicações IoT de forma rápida e barata. Em pouco tempo, milhares de desenvolvedores estão produzindo aplicações baseadas nesse microcontrolador para uma ampla gama de aplicações.

3.6.1 Módulos com o ESP8266

Há vários módulos disponíveis no mercado que usam o microcontrolador ESP8266. Eles podem ser encontrados em sites chineses, como Aliexpress

(*http://www.aliexpress.com*), Banggood (*http://www.banggood.com*), ou DealeXtreme(*http://www.dx.com*). Também podem ser localizados em sites brasileiros, como o Mercado Livre (*http://www.mercadolivre.com.br*), e sites especializados em componentes eletrônicos. Os preços partem de 1 dólar para a versão mais simples, e podem ser encontrados com opções de frete grátis acima de cinco unidades.

O módulo **ESP-01** é o módulo mais simples disponível. Além do microcontrolador ESP8266, ele tem apenas o módulo de memória externa, encontrado em versões de 512 kB ou 1 MB, suficiente para a maioria das aplicações. Tem apenas oito pinos, que incluem alimentação, comunicação serial UART e alimentação de 3,3 volts. Para ser gravado, é necessário o uso de adaptadores USB, disponíveis nos mesmos sites a baixo custo.

O ESP-01 é muito usado como módulo de comunicação WiFi para o Arduino e até para o Raspberry Pi, devido ao baixo custo. No entanto ele é capaz de realizar as tarefas de forma independente, visto que também tem processador interno com capacidade de processamento muito superior à da maioria dos módulos Arduino. Custa a partir de 1 dólar, mesmo preço pelo qual pode ser encontrado também um adaptador USB para sua gravação. É ideal para aplicações mais simples como acender uma lâmpada ou ler um sensor do tipo liga/desliga. A figura 3.4 apresenta uma imagem do ESP-01.

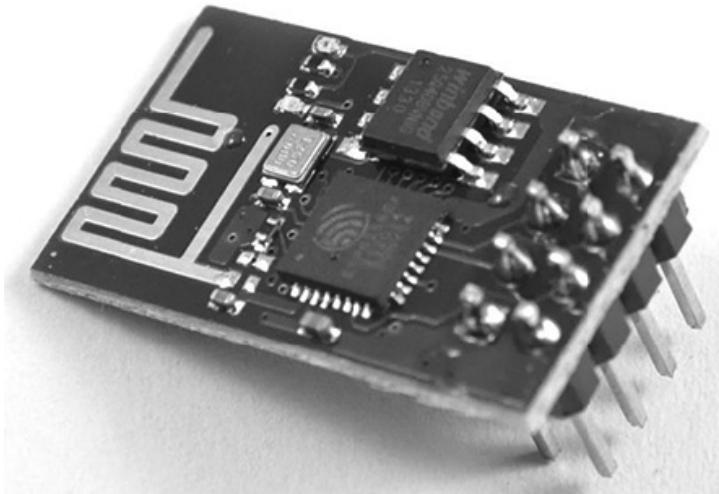



Figura 3.4 – Módulo ESP-01.

O próximo módulo disponível é o **ESP-12**, com suas variantes ESP12-E e ESP-12F, cuja diferença se localiza apenas na pinagem. Esse módulo foi construído para ser integrado a uma placa de circuito impresso que pode

agregar outros componentes. Vários outros módulos comerciais integram o ESP-12 em seus módulos. Tem 11 pinos GPIO, interfaces seriais SPI, I2C e UART. Contém memória flash de 4 MB e possibilita explorar todos os recursos do ESP8266. Pode ser encontrado nos sites chineses por menos de 2 dólares.

A figura 3.5 apresenta o ESP-12E. Ele não é tão fácil de usar na prototipação de projetos. Seu uso é mais direcionado na produção em média e larga escala, integrando as placas de circuito impresso que podem incluir outros componentes.

Uma alternativa ao ESP-12, com conector para antena, é o **ESP-07**. Por esse motivo, pode ser usado em aplicações que requerem antena externa. Os recursos, a pinagem e as aplicações são semelhantes aos do ESP-12. A figura 3.6 apresenta o ESP-07.

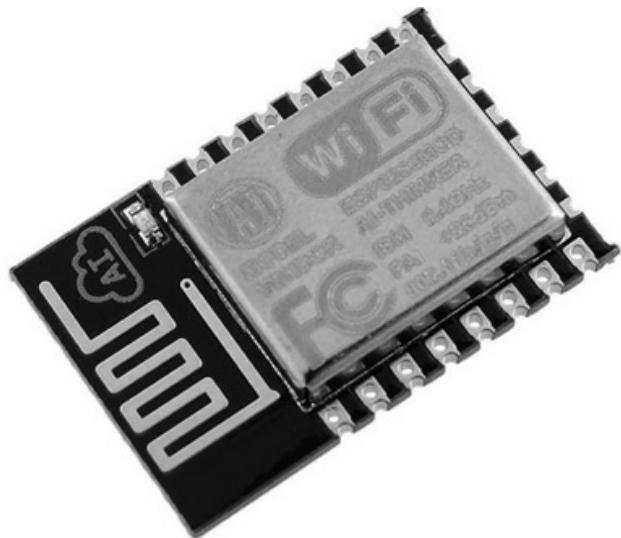



Figura 3.5 – Módulo ESP-12E.

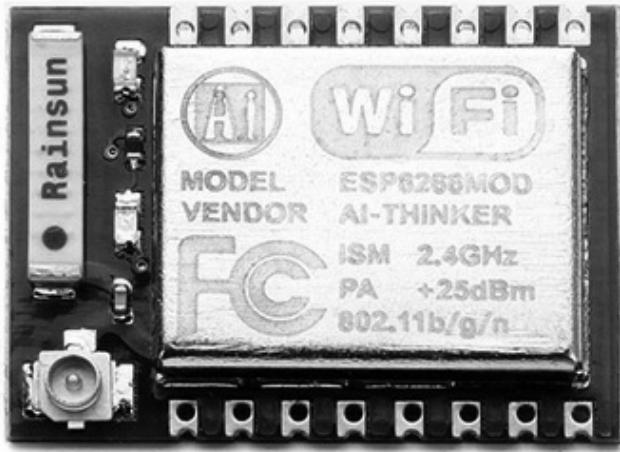



Figura 3.6 – Módulo ESP-07.

Os próximos módulos utilizam o ESP-12 em uma placa que agrega recursos adicionais, especialmente um conversor USB-serial, para facilitar a gravação do software e um regulador de tensão de 5 volts para 3,3 volts, o que possibilita ligar os módulos diretamente em interfaces USB, bem como em fontes de 5 volts, bem mais comuns que as fontes de 3,3 volts.

O módulo mais popular é o módulo chamado **NodeMCU**, que agrega com ele uma plataforma de desenvolvimento em linguagem Lua. Essa linguagem, desenvolvida no Brasil por professores da PUC-Rio, é interpretada e tem uma sintaxe de fácil aprendizagem e integração. A figura 3.7 apresenta o módulo NodeMCU em sua versão 3.0. Foram lançadas algumas versões, com algumas divergências na numeração, que basicamente se resumem ao ESP-12 com o conversor USB-serial e o regulador de tensão. A pinagem facilita a prototipação e a montagem de pequenos circuitos, sendo indicada para os leitores fazerem seus primeiros experimentos. Custa a partir de 3 dólares nos sites chineses e pode também ser encontrado com facilidade em sites brasileiros.

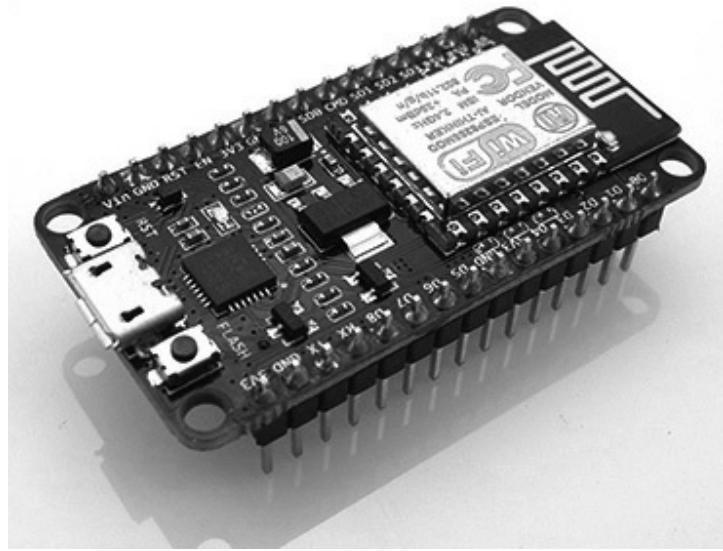



Figura 3.7 – Módulo NodeMCU.

Outro módulo bastante acessível e de fácil integração é o **Wemos D1 mini** (Figura 3.8), às vezes referido apenas como D1 mini. Também é construído com o ESP-12 e conta com conversor USB-serial e regulador de tensão. É compatível no âmbito de software com o NodeMCU, incluindo o suporte à linguagem Lua. Sua maior vantagem são as dimensões reduzidas, quase metade do tamanho da placa do NodeMCU. O tamanho também limita o número de portas GPIO, reduzidas para 8 portas, o que é suficiente para a maioria das aplicações. Também pode ser encontrado por 3 dólares nos sites chineses.

Figura 3.8 – Módulo Wemos D1 mini.

A partir deste capítulo, vamos considerar os módulos integrados do

microcontrolador ESP8622EX para o desenvolvimento de suas aplicações, exceto em aplicações específicas, nas quais serão exigidos recursos adicionais, como maior poder de processamento ou outras interfaces para integração de dispositivos periféricos. Nesse caso, a placa Raspberry Pi será usada com mais recursos e suporte a periféricos e software.

Os exemplos apresentados neste livro também podem ser integrados em um módulo Arduino típico, como Arduino Uno, Nano etc., desde que tenha um módulo adicional de rede WiFi. Esses módulos podem ser facilmente encontrados e já são de domínio de boa parte dos desenvolvedores Arduino. O próprio ESP8266 pode ser usado como interface WiFi no módulo ESP-01. É um claro desperdício de recursos, visto que o ESP8266 dá conta de todo o processamento, mas o Arduino pode ser necessário por causa de suas interfaces e portas.

CAPÍTULO 4

Gestão de energia em IoT

O consumo de energia é fundamental para o funcionamento dos dispositivos de Internet das Coisas. Consumo elevado inviabiliza o uso de baterias, essencial em algumas aplicações, ou até mesmo painéis solares. Há, ainda, que se considerar o custo do consumo de energia dos dispositivos IoT que estejam ligados constantemente à rede elétrica.

O consumo de energia dos dispositivos IoT pode ser de extrema importância, a depender da sua aplicação. Diversos cenários exigem um consumo baixo de energia, ou extremamente baixo, se forem mantidos por baterias sem possibilidade de recarga, ou com alto custo de recarga e substituição. Alguns ambientes, como residências, são tipicamente de baixo consumo porque são mantidos com orçamentos pessoais e, normalmente, restritos. Outros ambientes, como aplicações de monitoramento ambiental, simplesmente não contam com fonte de energia permanente e devem prever consumo extremamente baixo, ou se tornam inviáveis.

A fonte de energia primária, ou seja, a ser utilizada na maioria das aplicações, é a energia fornecida pelas empresas distribuidoras de energia, em tensão de 127 ou 220 volts, que pode ser facilmente convertida para corrente contínua, em tensão de 3,3 ou 5 volts, para alimentar os dispositivos IoT. Nesse caso, em geral, o impacto do consumo de energia é muito baixo, a menos que se faça uso de computadores pessoais na aplicação que podem apresentar um alto impacto se permanecerem ligados de forma constante.

Ambientes remotos, como em aplicações de monitoramento ambiental, por sua vez, não requerem uma fonte constante de energia, o que exige um consumo mínimo para que as fontes alternativas a ser consideradas, incluindo energia solar, eólica e até hidroelétrica, possam ser

dimensionadas sem pesar o orçamento da aplicação.

Dispositivos propostos para atuar em redes de sensores sem fio como o Telos B (*http://web.stanford.edu/class/cs240e/papers/spots05-telos.pdf*) chegam a apresentar consumo de 41 mW em modo ativo com rádio ligado, com tensão de alimentação mínima de 1,8 volt, sendo considerados como *ultra low power*. Esses dispositivos utilizam rádio no padrão IEEE 802.15.4, e o alcance do rádio se limita a poucos metros, restringindo suas aplicações.

Os dispositivos com redes WiFi requerem consumo, pelo menos, uma ordem de grandeza acima dos dispositivos que só comunicam nas redes-padrão IEEE 802.15.4. No entanto a maioria das aplicações suporta o consumo requerido. Apenas aplicações nas quais a fonte de energia não pode ser carregada, tampouco substituída, não suportariam os dispositivos referidos neste trabalho, como o ESP8266 e o Raspberry Pi, que têm interfaces WiFi.

4.1 Consumo dos dispositivos IoT

Graças à grande diversidade de dispositivos que podem ser usados em aplicações de IoT, é preciso classificá-los para compreender bem seu consumo de energia. Dispositivos IoT podem consumir tão pouco quanto os módulos projetados para redes de sensores sem fio, que consomem na ordem de 50 MW, em modo de transmissão; ou computadores pessoais do tipo desktop, que podem consumir 200 W, mesmo sem transmitir nada pela rede.

O consumo de um sistema microcontrolado depende de sua potência, que é produto de sua tensão *versus* a corrente. Dispositivos estão sendo construídos para funcionamento em tensões menores, exatamente para reduzir o consumo do sistema. Além disso, é preciso considerar o consumo dos módulos periféricos, sejam sensores, atuadores ou módulos de comunicação agregados.

Considerando um módulo microcontrolado com capacidade de comunicação sem fio, é possível comparar o consumo dos principais módulos que poderiam ser usados para implementar dispositivos IoT.

4.2 ESP8266

O consumo dos módulos com o ESP8266 é, basicamente, o consumo do microcontrolador, visto que este é o único componente ativo do módulo. Os demais componentes, como o conversor USB-serial e o redutor de tensão 5 V → 3,3 V, poderiam ser desligados.

O microcontrolador ESP8266, em seu modo de funcionamento normal, transmitindo com potência máxima de 17 dBm, consome 170 mA, que seria seu consumo máximo, sem contar ligações de dispositivos periféricos de I/O. No outro extremo, o modo *deep-sleep*, no qual todas as suas funções são desligadas por um período predeterminado, o consumo cai para 10 uA. Os demais modos e consumos são:

- **Transmitindo** – a 17 dBm, 170 mA; a 15 dBm, 140 mA; a 13 dBm, 120 mA.
- **Recebendo** – a -80 dBm, 50 mA; a -65 dBm, 56 mA.
- **Modo modem-sleep** – 15 mA. Neste modo, o WiFi é desligado entre os intervalos de envio do *beacon* pelo roteador. Somente funciona no modo *station*, sem realizar qualquer operação de recepção ou transmissão de pacotes. A CPU continua funcionando e pode operar as operações de I/O.
- **Modo light-sleep** – 0,5 mA. Nesse modo, a CPU também é desligada, mas pode ser acionada por um sinal de I/O. Normalmente, esse modo é usado em aplicações que só precisam agir na presença de um evento de I/O.
- **Modo deep-sleep** – 10 uA. Nesse modo, todos os circuitos são desligados por um determinado tempo indicado. Após esse tempo, a CPU reinicia, sendo necessários todos os procedimentos de conexão e configuração.

O modo *deep-sleep* tem consumo praticamente insignificante. Uma bateria de 1.000 mAh poderia manter esse modo por 100 mil horas, mais de 11 anos. Só não poderia realizar nenhuma tarefa nesse modo. Mas pode ser programado para acordar periodicamente, realizar uma tarefa, como ler um sensor e enviar pela rede, e depois voltar a dormir. Esse processo pode levar cerca de 5 segundos, visto que a CPU é reiniciada. Logo, para se justificar, o período de tempo no modo *deep-sleep* deve ser longo, da

ordem de minutos.

Para tarefas que devam ser respondidas de forma mais rápida, os modos *light-sleep* e *modem-sleep* podem ser mais interessantes.

4.3 Raspberry Pi

O módulo Raspberry Pi não é econômico como o ESP8266. Ele está algumas ordens de grandeza acima. No entanto, comparado com computadores pessoais, ainda pode ser uma opção econômica e viável. Há várias versões disponíveis, com diferentes modos de consumo. Além disso, apesar de um grande número de periféricos incorporados à placa, como saída HDMI, esta não tem a comunicação WiFi incorporada, exceto na versão 3. A ligação de um adaptador WiFi USB pode, ainda, aumentar o consumo.

Inicialmente lançado nas versões A+ e B+, diferenciando-se pelo número de periféricos, os módulos Raspberry Pi ganharam novas versões – 2 e 3 –, aumentando o poder de processamento e os periféricos incluídos. A versão 3 conta com WiFi e Bluetooth incorporados, possibilitando, assim, diversas aplicações sem acréscimo de hardware e consequente consumo adicional.

A tabela 4.1 apresenta o consumo das versões do Raspberry em situações distintas: inicialização, ociosa, execução de vídeo e sobrecarga de processamento. O consumo varia de 100 mA a 1.340 mA. Caso seja necessária a inclusão de um adaptador WiFi, é preciso considerar cerca de 40 mA adicionais apenas para sua alimentação e mais 100 mA em modo de transmissão.

Tabela 4.1 – Consumo das placas Raspberry Pi

Consumo (mA)	Pi1 (B+)	Pi2 B	Pi3 B	Zero
Inicialização	220	220	350	150
Ocioso	200	220	300	100
Vídeo	220	280	330	160
Sobrecarga	350	820	1340	350

Considerando um consumo médio de 400 mA a uma tensão de entrada de 5 volts, tem-se uma potência média de 2 W. Se mantido por baterias, a

autonomia em bateria é baixa, se aproximando da autonomia de smartphones e dificilmente ultrapassando um dia. E com um problema adicional de utilizar a tensão de 5 volts, tensão na qual é difícil encontrar baterias. As baterias disponíveis em 5 v são, normalmente, baterias reservas para proporcionar carga extra em smartphones. Estes, por sua vez, funcionam, normalmente, com baterias de 3,7 volts, tensão bem mais comum em baterias.

O consumo do Raspberry ligado à rede elétrica de forma constante pode ser, também, considerado. Com 2 W médios, seu consumo mensal é de 1,44 KWh, o que é, relativamente, insignificante, mesmo para consumo residencial, o que habilita sua utilização sem grandes impactos para esse cenário, inferior a um roteador WiFi ou receptor de TV por satélite.

Uma alternativa ao Raspberry Pi, o módulo conhecido como Beagleboard, tem consumo e desempenhos similares aos do Raspberry, mas uma vantagem importante quanto ao gerenciamento de energia: ele é capaz de manter seu funcionamento com a tensão mínima de 3,5 volts, o que o habilita a funcionar com as baterias de 3,7 volts, disponíveis em diversos modelos e tecnologias, como íons de lítio e polímero de lítio, que têm as melhores relações entre volume, capacidade, peso e preço.

A tabela 4.2 apresenta um comparativo entre o consumo dos principais módulos para IoT, comparado com soluções de consumo extremamente baixo (Telos Motes) e BLE nRF51822, smartphones e laptops. O módulo Telos Motes foi proposto para ser um nó em uma rede de sensores. Tem um processador mais limitado, rádio de poucos kbps e alcance de poucos metros, e foi proposto para aplicações simples de sensoriamento e repasse de mensagens. O nRF51822 é um módulo bastante adequado para IoT, mas tem um rádio bem mais limitado, chegando até 4 dBm de potência de transmissão, muito inferior aos 17 dBm do ESP8266. Como parâmetro, é possível perceber que o ESP8266 não está muito longe em termos de consumo. O Arduino Nano, uma versão do Arduino com hardware bem mais simples, sem recursos de rede, tem consumo superior ao ESP8266, como pode ser visto na tabela. O Raspberry Pi, por sua vez, tem consumo muito superior, visto que não é um módulo de baixo consumo, apesar de os números serem, ainda, muito inferiores aos de um laptop, ao qual o poder de processamento é comparável.

Tabela 4.2 – Comparativo de consumo de diversos dispositivos portáteis

Módulo	Consumo			
	Sleep	Sem comunicação	Recebendo	Transmitindo
Telos Motes	15 µW	3 mW	41 mW	41 mW (0 dBm)
BLE nRF51822	1,8 µW	7 mW	39 mW	31,4 mW (0 dBm)
ESP8266	33 µW	49,5 mW	165 mW 184,8 mW	561 mW (17 dBm) 462 mW (15 dBm) 396 mW (13 dBm)
Arduino Nano	115 µW	75 mW		
Raspberry Pi	–	1.150 mW 1.500 mW 1.650 mW		
Smartphone	150 mW	500 mW	750 mW	1.000 mW
Notebook	500 mW	14 W	14 W	14 W

4.4 Baterias

O desenvolvimento de baterias cada vez mais eficientes é chave para diversas tecnologias emergentes, como carros elétricos e drones. O desenvolvimento das baterias também é útil para aplicações IoT, visto que dão mais autonomia para as aplicações que não podem ser ligadas à rede elétrica.

Diversos tipos de baterias estão disponíveis, com grande variedade de preço, volume, peso e capacidade. Entre as baterias comerciais, as melhores relações de peso e capacidade são as das baterias de lítio, sejam de íons de lítio ou de polímero de lítio. Entre esses dois tipos, os polímeros de lítio ou LiPo produzem baterias mais leves e flexíveis, porém estas são mais caras. As baterias de íons de lítio são mais baratas e têm relações parecidas de capacidade por volume e tempo de vida. Ambas não sofrem com efeito memória e podem ser carregadas e descarregadas em qualquer nível de carga sem danificar seu tempo de vida.

As baterias de polímeros de lítio, por serem mais leves, são usadas em drones, além dos equipamentos da Apple: iPhone e iPad. As baterias de íons de lítio são amplamente usadas pelos demais fabricantes de smartphones, bem como em laptops e outros dispositivos portáteis.

As células básicas das baterias de lítio têm tensão de 3,7 volts. Assim, elas podem ser encontradas nesses valores e seus múltiplos: 7,4 volts, 11,1 volts e 14,8 volts, usando associação em série. Os telefones celulares, smartphones e tablets normalmente usam apenas uma célula de 3,7 volts sendo alimentada por carregadores de 5 volts. Como essa é a tensão das portas USB, eles podem ser facilmente carregados em qualquer dispositivo que tenha uma porta USB, e isso tem incentivado a sua popularização.

As baterias de níquel-metal-hidreto (NiMH) são também amplamente difundidas e comerciais. Têm boa capacidade por volume e peso, e a tensão das células é de 1,2 volt, podendo ser ligadas em série para obter 2,4 volts, 3,6 volts e 4,8 volts. São normalmente usadas em formato de pilhas AA e AAA, com o propósito de substituir essas pilhas, não recarregáveis.

Configurações de 5 e 12 volts também estão disponíveis, especialmente como alternativas para carregar telefones celulares e substituir fontes que utilizam essas tensões. Para cada aplicação é preciso identificar o melhor tipo de bateria, observando relações de peso e volume, capacidade e custo. Cada aplicação apresenta requisitos específicos que podem demandar um determinado tipo de bateria.

Baterias chamadas de metal-ar estão em franco desenvolvimento, especialmente para os carros elétricos, e apresentam relações excelentes de peso e volume por capacidade. Utilizam o oxigênio disponível no ar que respiramos como um dos elementos da bateria, por isso ficam mais leves e compactas. Poderão ser intensamente usadas, no futuro, especialmente as baterias de lítio-ar.

Além das baterias recarregáveis, também é possível usar baterias descartáveis, de carga única, como as baterias CR2032, usadas em placas-mãe de computadores pessoais, bem como controles remotos e similares. Sua capacidade é de cerca de 200 mAh.

4.4.1 Cálculo da autonomia da bateria

A escolha da bateria deve considerar a tensão necessária e a autonomia requerida para a aplicação. O ESP8266 utiliza alimentação de 3,3 volts. Algumas versões do Arduino também utilizam 3,3 volts. Outras versões utilizam 5 volts, bem como o Raspberry Pi. O módulo BeagleBone,

concorrente do Raspberry, também utiliza 5 volts, mas tem uma tolerância, funcionando na tensão mínima de 3,4 volts, o que possibilita utilizar bateria de uma célula de lítio para alimentá-lo.

Após a escolha da tensão da bateria, a autonomia será dada pela divisão da capacidade da bateria, dada em mAh, pelo consumo médio. O consumo médio deve considerar a proporção do tempo em que os módulos permanecem em cada estado, seja de transmissão, recepção ou *sleep*, se disponível.

O ESP8266 tem tensão de trabalho entre 3 e 3,6 volts. Para tanto, uma bateria de uma célula de lítio, com 3,7 volts, pode ser usada com um regulador de tensão de 3,3 volts. Essas baterias são encontradas em inúmeras configurações de capacidade na ordem de milhares de mAh. Considerando uma bateria de 2000 mAh, e um consumo médio de 40 mA no ESP8266, a autonomia dessa bateria seria de 50 horas. O consumo médio pode ser reduzido, mantendo o ESP8266 durante mais tempo em modo *sleep* e só ativando o módulo periodicamente para executar as tarefas necessárias. É possível estender a autonomia por um período longo, possivelmente até anos, lembrando que o modo *deep-sleep* tem consumo de apenas 10 uA. Nesse modo, somente, a autonomia seria de 200 mil horas, o correspondente a 22 anos sem recarregar ou substituir as baterias. Dificilmente esse cenário seria possível pela ação do tempo sobre as baterias. Elas não manteriam a carga por tanto tempo, mesmo sem utilização.

O dimensionamento da bateria deve considerar quanto tempo se espera manter o dispositivo ligado sem recarregar ou substituir as baterias, o que depende da aplicação, da possibilidade de recarga e da substituição das baterias. A capacidade da bateria é dada em Ah ou seus múltiplos, sendo comum o mAh, sua milésima parte. A tensão da bateria deve ser escolhida de acordo com a tensão de entrada do dispositivo, verificando-se os limites máximos e mínimos. Após escolhida a bateria, sua autonomia, em horas, pode ser obtida dividindo a capacidade da bateria pela corrente média do dispositivo.

4.5 Soluções sustentáveis com painel solar

Painéis solares fotovoltaicos se tornaram populares nos últimos anos, possibilitando sua utilização em aplicações sustentáveis diversas, com destaque para aplicações de microeletrônica que têm baixo consumo e aplicabilidade em diversos setores. Para o cenário de IoT, a alimentação por energia solar abre a perspectiva de novas aplicações, especialmente em cenários remotos e de difícil acesso.

Há vários tipos de painéis solares, sendo mais comuns os monocristalinos e policristalinos, que podem ser vistos na figura 4.1. Os painéis monocristalinos são mais eficientes, gerando mais energia por uma área semelhante, mas também tendem a ser um pouco mais caros. São melhores em situação de pouca luz e devem ser preferidos em aplicações com restrições de espaço, peso e luminosidade.

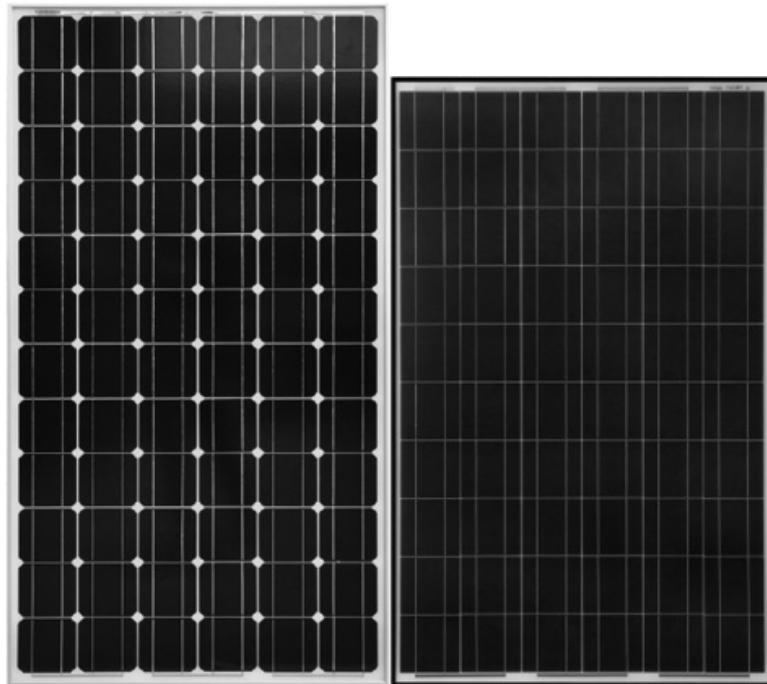



Figura 4.1 – Painel solar monocristalino (a) e policristalino (b).

O dimensionamento de painéis solares para aplicações IoT deve considerar a capacidade da bateria utilizada, a disponibilidade desejada e a capacidade solarimétrica da região. O Atlas Solarimétrico do Brasil (*http://www.cresesb.cepel.br/publicacoes/download/Atlas_Solarimetrico_do_B* apresenta informações aproximadas de radiação solar em todo o Brasil. A média fica em torno de 6 a 7 horas diárias de sol, com destaque positivo para o Nordeste, chegando a 8 horas de média diária no sertão nordestino,

e negativa para a região norte. Algumas áreas apresentam apenas 4 horas médias de sol por dia. A figura 4.2 mostra as médias anuais.

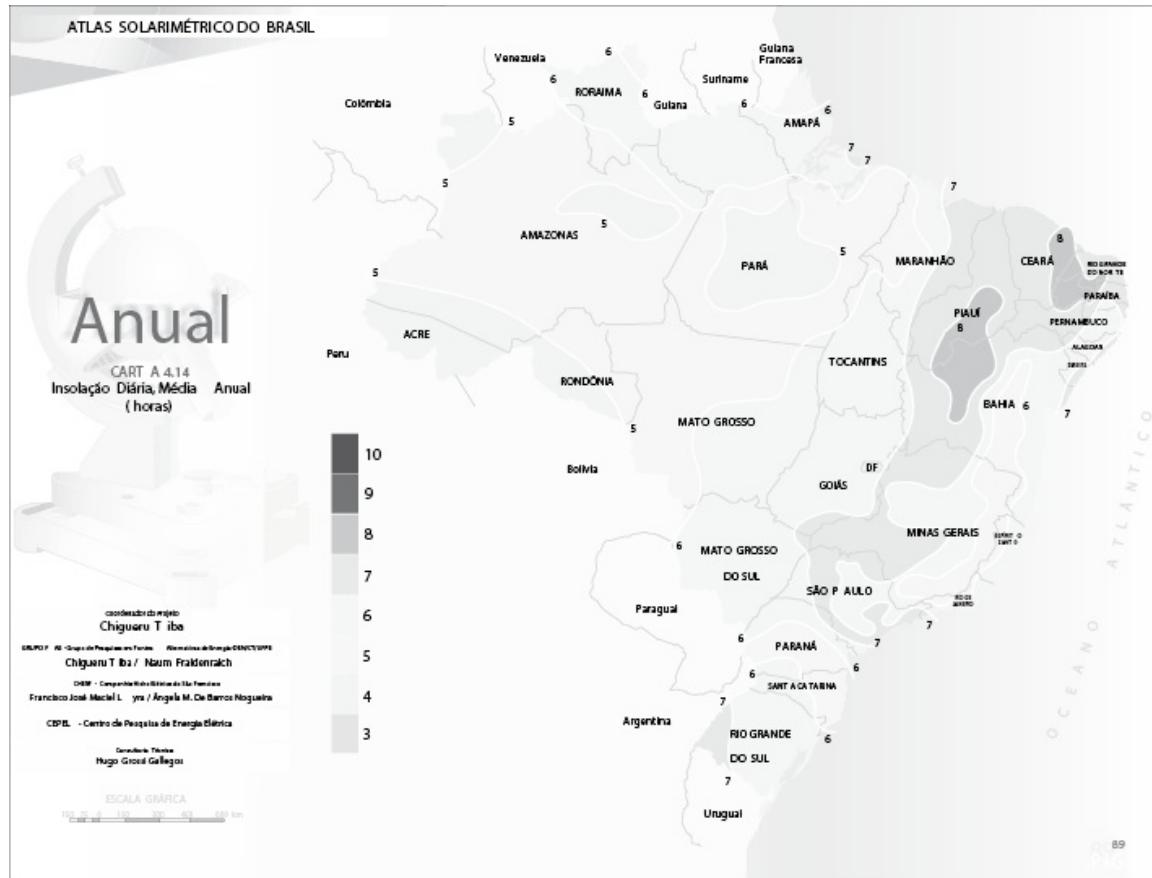



Figura 4.2 – Média de insolação diária no Brasil.

Para dimensionar painéis solares para qualquer sistema é preciso considerar o consumo nominal diário, a geração nominal diária e o tempo desejado para carregar totalmente as baterias. Subtraindo-se o consumo da geração nominal diária, obtém-se a energia a ser armazenada diariamente. Esse valor deve ser suficiente para carregar a bateria e deve ser proporcional ao tempo que se espera para atingir a carga total da bateria.

Considere um exemplo típico um módulo com o ESP8266 que tenha consumo médio de 50 mA, que se queira manter em funcionamento de forma ininterrupta. Dimensionar a bateria para mantê-lo ligado por cinco dias é suficiente para garantir uma alta disponibilidade, acima de 99%, para a maioria das regiões brasileiras. Cinco dias representam 120 horas; com consumo médio de 50 mA, obtém-se o valor de 6.000 mAh para o conjunto de baterias, que podem ser de lítio, com 3,7 volts.

O tempo de vida de uma bateria depende diretamente do percentual de descarga. Baterias que se mantêm entre 80% e 100% podem ter seu tempo de vida até cinco vezes maior que aquelas que descarregam até 20% da sua carga. Essa é mais uma razão para dimensionar as baterias para períodos longos. No caso de recarga solar, dimensionando a bateria para se manter por cinco dias, já se projeta o sistema para utilizar, em média, apenas 20% da carga. A descarga será superior apenas para os dias em que o sol não aparecer. E o caso extremo, de carga mínima, somente ocorrerá em situações extremas.

Para o cálculo do painel solar, deve-se considerar o consumo diário nominal, de 50 mA *x* 24 h = 1.200 mAh. Esse valor é suficiente para manter o funcionamento do dispositivo durante 24 horas, o que indica seu funcionamento durante o período de sol, além da carga da bateria nesse período para o período sem sol. Além disso, é preciso prever uma carga adicional da bateria que possa mantê-la constantemente carregada para manutenção da energia nos dias sem sol. Um adicional de 50% para a carga adicional deve ser suficiente para garantir uma alta disponibilidade do sistema, resultando em 1.800 mAh, valor que será considerado no dimensionamento do painel solar.

O painel solar é especificado com tensão e potência. Para carregar baterias de 3,7 volts, um painel com tensão de 5 volts é suficiente. A potência necessária será calculada a partir da corrente desejada, 1.800 mAh diários *x* a tensão, o que resulta em 9 Wh. Considerando uma média de 6 horas de sol para a região em que será instalado, um painel solar de 1,5 W seria suficiente para essa aplicação.

4.6 Controlador de carga

Para prolongar o tempo de vida da bateria é preciso estabelecer limite máximo de carga e limite mínimo de operação. O limite máximo de carga tem o objetivo de evitar uma sobrecarga da bateria, o que pode até causar uma explosão desta, especialmente das baterias de lítio. O limite mínimo de operação visa proteger a bateria contra a descarga total, que pode danificar as placas da bateria, reduzindo, significativamente, seu tempo de vida.

Visando eliminar a sobrecarga, bem como a descarga abaixo de 20%, é necessária a utilização de um circuito controlador de carga. Há várias opções disponíveis no mercado para grande variedade de capacidades de baterias e painéis solares; em geral, para painéis de grande potência e baterias de maior capacidade, mas é possível encontrar alguns apropriados para aplicações IoT, com apenas uma célula de lítio, além de baixa corrente e custo acessível.

Um controlador de carga tem entrada para o painel solar, e saída para a bateria e a carga, que será composta com o dispositivo IoT e seus periféricos. Vale lembrar que os periféricos, como módulos sensores e atuadores, podem consumir até mais que o módulo de processamento e comunicação.

O controlador de carga pode atuar na recarga da bateria mesmo que esta não seja feita por meio de um painel solar, possivelmente de uma fonte de alimentação ligada à rede elétrica. Nessa função, sua presença também é importante para evitar sobrecarga e exaustão total da bateria, que podem levar à redução de sua vida útil.

A figura 4.3 apresenta uma placa de controlador de carga de baixa potência que pode ser usada em dispositivos IoT, como os módulos com o microcontrolador ESP8266. Essas placas custam a partir de 2 reais em sites chineses, como o Aliexpress, ou a partir de 3,50 reais em sites brasileiros como o Mercado Livre. Para localizar, basta pesquisar por “carregador USB para bateria de lítio”.

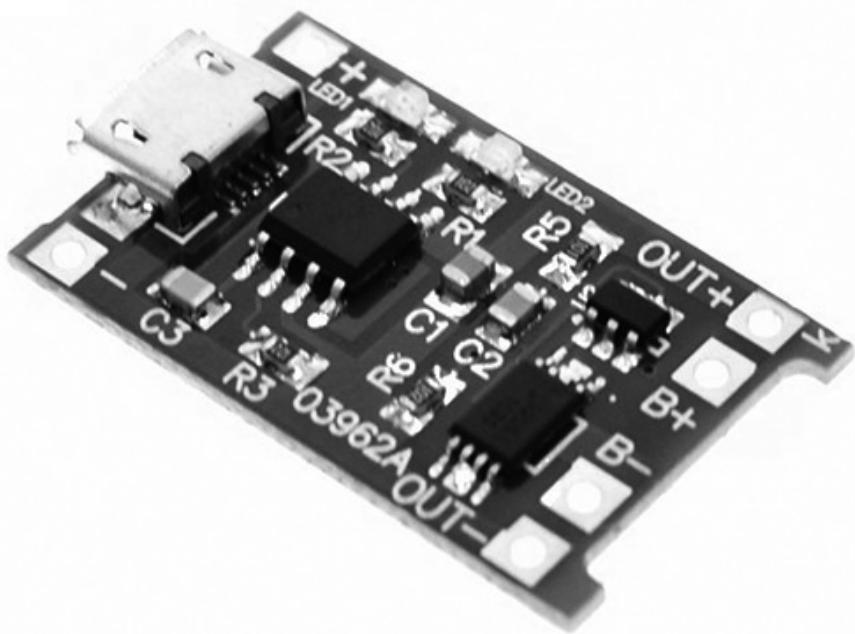



Figura 4.3 – Placa de controlador de carga de baixa potência.

As aplicações com restrições de energia, ou ainda, aplicações sustentáveis, são cada vez mais comuns no cenário de Internet das Coisas. É importante considerar a escolha correta do microcontrolador, bem como dimensionar o tempo em que ele deve permanecer no modo sleep, em modo de economia de energia. Aplicações típicas de baixo consumo de energia podem se manter no modo *sleep* por mais de 99% do tempo, economizando sua fonte de energia.

4.7 Exemplo de aplicação autossustentável

Para efeito de ilustração, considere-se um módulo com ESP8266, que permaneça 95% no modo *deep sleep* (33 µW), e 3% no modo ativo (49,5 mW), e 2% (396 mW) no modo transmitindo, com potência de 13 dBm. A média ponderada pode ser calculada por 0,95 x 0,033 + 0,03 x 49,5 + 0,02 x 396 = 9,44 mW que, a 3,3 volts, dá uma corrente média de 2,86 mA. Uma bateria de íons de lítio de 2.500 mAh e 3,7 volts poderia mantê-lo em funcionamento por 874 horas, o que corresponde a 36 dias.

Nessa situação, o consumo diário seria de 69 mAh. Com uma insolação média de 6 horas por dia, comum na maior parte do Brasil, um painel solar com geração de 11,5 mA já atenderia à demanda. Para carregar uma bateria de 3,7 volts, a tensão do painel de 5 volts é suficiente. Assim, um

micropainel de 5 volts e 30 mA, como o da figura 4.4, seria suficiente para suprir a demanda. Esse micropainel custa a partir de 1 dólar nos sites chineses.

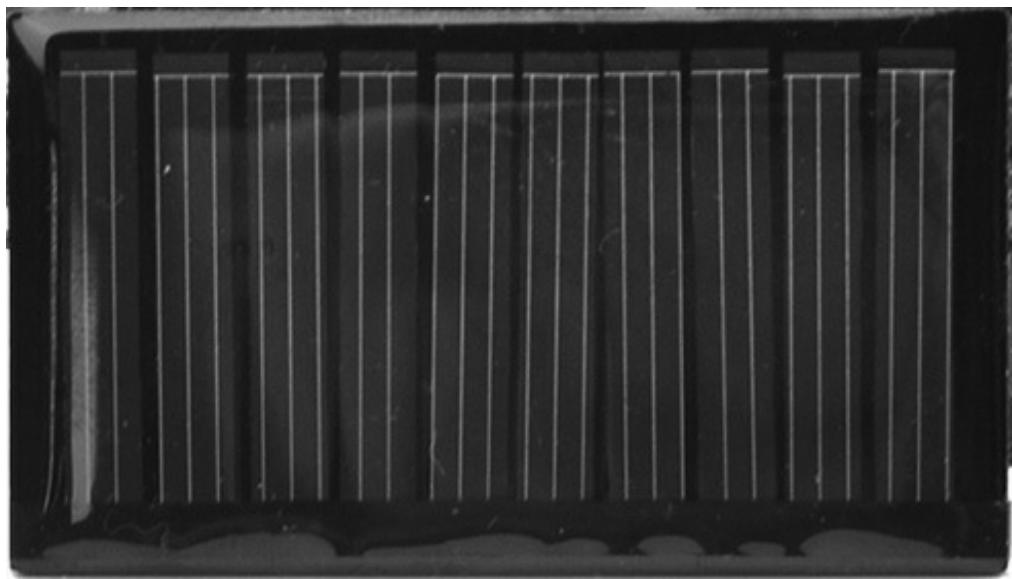



Figura 4.4 – Micropainel solar de 30 mA.

O micropainel tem dimensões de 53 mm x 30 mm e seria o maior componente desse sistema, optando por usar o ESP-12, com dimensões de 24 mm x 16 mm, um controlador de carga de 26 mm x 17 mm e a bateria de 31 mm de comprimento e diâmetro de 16 mm. Ou seja, cabe tudo em uma caixinha de fósforos.

O custo da solução vai girar em torno de 2 dólares para o ESP12, mais 1 dólar para o micropainel, 1,5 dólar para a bateria e 50 centavos de dólar para o controlador de carga de baixa potência, resultando em um sistema autossustentável de 5 dólares. Esse preço não inclui taxas de importação, que podem dobrar o custo dos produtos nos estados que cobram ICMS sobre as importações, ou aumentar 60% nos estados que isentam ICMS sobre esse tipo de compra. Por outro lado, compras em grandes quantidades podem baratear a compra desses componentes.

Ainda é possível reduzir o consumo do sistema deixando o microcontrolador no modo *deep sleep* por até 99% do tempo, em aplicações como telemetria, em que o ciclo de leitura e transmissão possa ocorrer em intervalos de cerca de cinco minutos. A autonomia da bateria poderia aumentar até cinco vezes, durando meses sem recarga.

Aplicações autossustentáveis, mantidas por painéis solares, devem considerar o dimensionamento adequado do painel, bem como o controlador de carga que suporte a potência necessária e a bateria para manter o funcionamento durante o tempo necessário e sem perder grande parte da sua carga. O tempo de vida da bateria, nesse caso, pode comprometer a aplicação. Para prolongar o tempo de vida, esta deve manter sempre uma boa carga residual. O ideal seria que a carga nunca ficasse abaixo de 50%. No exemplo citado, o consumo diário deve ser menor que 3% da carga da bateria. Ou seja, mesmo em uma sequência de cinco dias sem sol, a carga da bateria não seria menor que 85%, o que garantiria sua vida útil por vários anos.

CAPÍTULO 5

Programação paralela, distribuída e de tempo real

Os princípios de programação para Internet das Coisas precisam considerar a concorrência no tratamento de eventos, o sincronismo entre os dispositivos, bem como as restrições temporais. Para tanto, a programação deve considerar mecanismos de sincronismo, bem como deve estar sempre pronta para responder aos eventos.

5.1 Programação paralela

Com o grande poder computacional dos sistemas modernos, os microcontroladores se tornaram capazes de executar várias tarefas ao mesmo tempo. Chamamos esse conceito de **multitarefa**. É esse princípio que permite que um usuário mantenha um navegador aberto ao mesmo tempo que digita em um editor de textos. O mesmo se espera dos microcontroladores, apesar do seu poder computacional ser inferior. Microcontroladores modernos podem ser capazes de controlar vários processos, se comunicar com outros dispositivos, salvar e carregar dados, em várias tarefas paralelas.

Os microprocessadores modernos já contêm **vários núcleos** e são capazes de, efetivamente, executar várias tarefas ao mesmo tempo. O mesmo não acontece com a maioria dos microcontroladores. Os vários núcleos melhoram o desempenho de sistemas multitarefas, mas, na prática, o conceito existe muito antes do conceito de processadores com vários núcleos. E este também vai se aplicar aos microcontroladores de único núcleo.

O conceito de multitarefa foi incluído nos principais **sistemas operacionais**. Cada programa em execução é chamado de processo. O

gerenciamento dos processos no computador cabe ao sistema operacional, de forma a decidir qual processo deve executar em cada momento de tempo. É possível, ainda, que um mesmo programa execute em duas instâncias diferentes. Nesse caso, seriam dois processos diferentes, apesar de serem instâncias do mesmo programa.

Os sistemas operacionais multitarefa podem ser **cooperativos** ou **preemptivos**. Nos sistemas cooperativos, é o próprio programa quem decide quando retornar o controle para o sistema operacional, o que normalmente ocorre após a conclusão de um procedimento específico. Nos sistemas preemptivos, é o sistema operacional quem decide quando deve interromper a execução de uma tarefa e passar a vez para outra.

O chaveamento de uma tarefa para outra normalmente é feito de forma temporizada, de modo que cada tarefa permaneça com o controle por um breve período de tempo, após o qual deixa de executar e passar ao final da fila. Ao retomar sua execução, esta tarefa deve se iniciar exatamente no mesmo ponto e com a mesma configuração de quando parou. Para que isso seja possível, o sistema operacional deve salvar o **contexto do processador**, que inclui o conjunto de registradores, o contador de programa e as demais informações essenciais para a retomada posterior do processo.

A figura 5.1 apresenta os vários processos em execução no sistema operacional Windows.

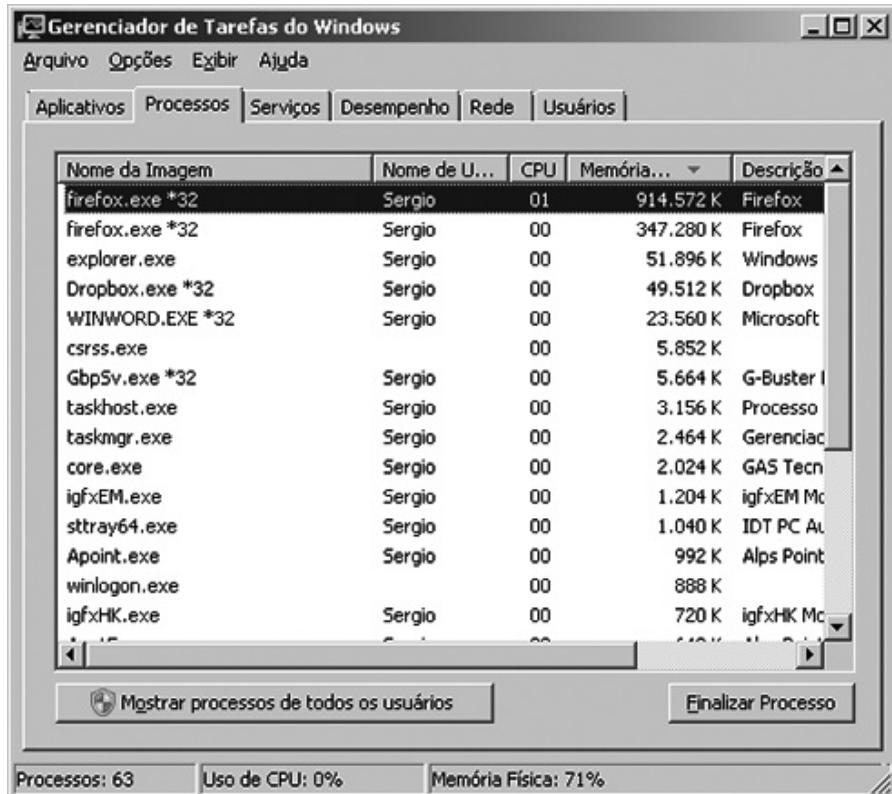



Figura 5.1 – Processos em execução no Windows.

Além de possibilitar que vários processos executem ao mesmo tempo, também é possível que um único programa conte com várias tarefas executando de forma concorrente, ao mesmo tempo. Assim, quando um navegador web possibilita abrir várias abas ao mesmo tempo, estas executam também de forma concorrente. Esse conceito de várias tarefas dentro de um mesmo processo é chamado de **thread**.

Eventualmente, as várias tarefas em execução em um computador, sejam processos ou threads, podem ter de se comunicar, seja para acessar informações comuns ou para trocar informações entre si. Como exemplo, já pensando em Internet das Coisas, podemos imaginar um dispositivo IoT controlando um sensor de temperatura que execute duas tarefas: a primeira tarefa lê a informação de temperatura e salva em uma tabela; a segunda tarefa recebe solicitações da rede, seja web ou qualquer outra forma, e disponibiliza dados dessa tabela. Ambas as tarefas teriam de acessar a mesma tabela. Quando duas tarefas compartilham informações ou dividem funções, é necessário usar conceitos de **Programação Paralela** para garantir que o fluxo de informações mantenha a integridade do sistema.

Embora existam sistemas operacionais específicos para microcontroladores, como o RTOS, sistemas microcontrolados normalmente usam um único programa, em bloco monolítico, que ocupa o núcleo único em 100% do tempo. Isso não impede que o programa tenha algum tipo de mecanismo de distribuição de tempo entre as várias tarefas para as quais é responsável. O código a seguir mostra uma divisão de tempo entre três tarefas em um código Arduino.

```
// Registro de execução das tarefas
unsigned long anterior_tarefa1 = 0;
unsigned long anterior_tarefa2 = 0;
unsigned long anterior_tarefa3 = 0;

// Definição de tempo das tarefas
const int tempo_tarefa1 = 1000;
const int tempo_tarefa2 = 1000;
const int tempo_tarefa3 = 1000;

void tarefa1() {
}

void tarefa2() {
}

void tarefa3() {
}

void loop() {
    if (millis() - anterior_tarefa1 > tempo_tarefa1) {
        tarefa1();
        anterior_tarefa1 = millis();
    }
    if (millis() - anterior_tarefa2 > tempo_tarefa2) {
        tarefa2();
        anterior_tarefa2 = millis();
    }
    if (millis() - anterior_tarefa3 > tempo_tarefa3) {
        tarefa3();
        anterior_tarefa3 = millis();
```

```
    }
  }
```

Neste código, cada uma das três tarefas é definida em uma função específica – (`tarefa1()`, `tarefa2()` e `tarefa3()`) – e está prevista para executar uma vez a cada 1.000 ms. A função `millis()` retorna o tempo, em milissegundos, desde o início da execução do microcontrolador. O problema dessa abordagem é que o tempo total de execução das tarefas não pode ultrapassar os 1.000 ms, ou a execução delas ficará atrasada. Nesse caso, não existe programação concorrente, visto que as tarefas devem ser totalmente executadas antes de passar o controle para a próxima tarefa.

Mecanismos simples de execução de várias tarefas, como este apresentado, são usados com frequência. Porém, devem ser usados com cuidado. Nenhuma instrução de espera ocupada, que mantém o processador parado na instrução, deve ser executada nas tarefas; caso contrário, comprometerá as demais tarefas. Assim, funções como `delay()` não devem ser usadas nas tarefas, nem mesmo funções que esperem resposta pela rede ou de um módulo, sensor ou qualquer outro dispositivo.

5.2 Programação distribuída

As tarefas executadas em rede, como acessar um site, ou acessar um banco de dados, dependem do fluxo de execução dos processos nos dois computadores. É um caso específico de execução paralela, no qual existem dois ou mais processadores responsáveis por um único fluxo de execução. A figura 5.2 apresenta uma situação típica de rede, na qual um cliente HTTP (web) solicita uma página a um servidor web. Essa página é recebida e contém um formulário que, depois de preenchido, é submetido novamente ao servidor. Este, por sua vez, pode acessar um servidor de banco de dados para buscar informações relevantes para o atendimento da solicitação do cliente. O servidor de banco de dados recebe a consulta, processa e a responde para, então, o servidor web ser capaz de responder ao cliente.

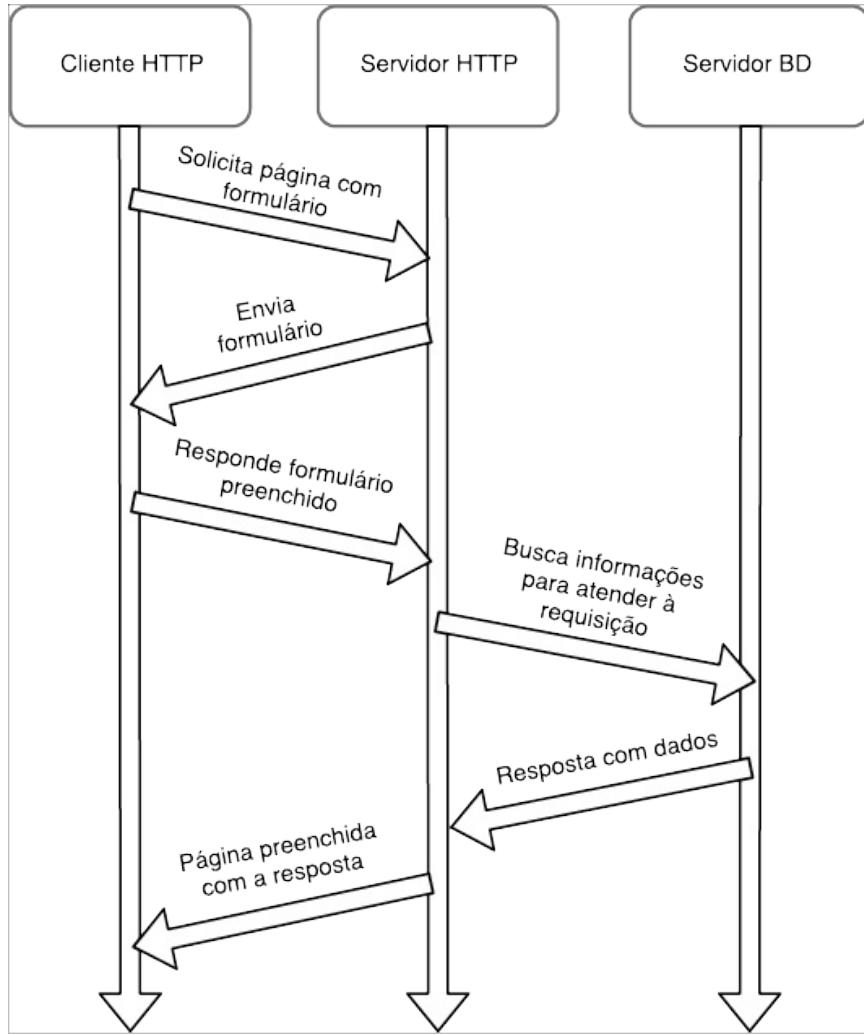



Figura 5.2 – Interações entre processos em diferentes computadores.

Em aplicações típicas em sistemas operacionais, uma tarefa ou thread específica é criada para tratar cada comunicação de rede. Cada uma das tarefas nos devidos elementos de rede poderiam esperar pela resposta do seu correspondente em uma espera ocupada. Em sistemas monolíticos executados nos microcontroladores, isso não é possível, visto que a espera ocupada pode comprometer as demais tarefas do sistema. Soluções alternativas podem ser usadas para ajudar a resolver esse problema.

O ambiente de desenvolvimento em Lua para o microcontrolador ESP8266, conhecido como NodeMCU, tem um esquema de funções para chamadas de retorno, ou *callback functions*. Essas funções são chamadas quando um evento, previamente informado, ocorre. Pode ser um evento de rede, como um pedido de conexão, ou um evento de I/O. As funções callback podem guardar o estado de um fluxo de comunicação, facilitando

a implementação de protocolos de comunicação.

5.3 Sistemas de tempo real

Sistemas microprocessados podem incluir atrasos na resposta a eventos muito além daqueles obtidos em sistemas eletrônicos não microprocessados. Os atrasos são provocados pelo sequenciamento das instruções, além dos tempos de espera incluídos entre procedimentos. Enquanto um sistema controlado por uma eletrônica não microprocessada é da ordem de microssegundos, os sistemas microprocessados tendem a incluir atrasos da ordem de centenas de milissegundos.

Alguns sistemas automatizados, entretanto, não suportam esse tipo de atraso. Uma linha de produção, por exemplo, em que sensores detectam a presença de material para usinagem, se tiverem atrasos da ordem de centenas de milissegundos, implicará em erros grosseiros na usinagem da peça, possivelmente tornando-a imprópria para utilização. Sistemas que têm restrições relativas ao atraso máximo a que podem estar sujeitos são chamados **sistemas de tempo real**.

Qualquer sistema que tenha restrição de tempo deve ser considerado sistema de tempo real. Em alguns casos, a restrição de tempo pode ser da ordem de minutos. Em outros casos, pode ser da ordem de microssegundos. Sistemas de tempo real com restrições rigorosas de tempo, os quais não podem ser ultrapassados em nenhuma hipótese, são chamados **sistemas de tempo real crítico**. Os sistemas de tempo real não críticos toleram que as restrições de tempo não sejam cumpridas, mas apenas com uma dada probabilidade, normalmente pequena. Entre os sistemas críticos podemos citar freios ABS, linhas de usinagem industrial, sistemas de piloto automático de aviões e automóveis.

Os microcontroladores apresentam maior demanda para atuar em sistemas de tempo real, visto que são mais usados em sistemas embarcados, responsáveis por comandar máquinas com restrições de tempo. Por esse motivo, o projeto de sistemas de tempo real é mais comum para microcontroladores. Existem vários sistemas operacionais de tempo real para microcontroladores. Um **sistema operacional de tempo real** é capaz de gerenciar tarefas com restrições de tempo, priorizando sua execução

dentro do tempo designado. Vários sistemas operacionais de tempo real estão disponíveis, alguns comerciais, como QNX e VxWorks, outros gratuitos, como o BRTOS, projeto brasileiro da Universidade Federal de Santa Maria (RS); e o FreeRTOS, disponível para o ESP8266.

Sistemas monolíticos com pseudotarefas temporizadas, como as mostradas na seção 5.1, também podem ser desenvolvidos de forma a atender restrições críticas de tempo. Para isso é necessária a utilização do mecanismo de interrupção do processador.

Interrupção é um mecanismo com origem no hardware que possibilita ao processador interromper o fluxo normal de instruções para atender a um evento. O evento pode ser originado por interfaces de I/O, temporizadores ou após a conclusão de determinadas tarefas, como comunicação ou gravação em memória permanente.

Ao identificar a ocorrência de uma interrupção, o processador para sua execução, salva o contexto atual, que pode incluir registradores, bem como o contador de programa, que indica o ponto onde o programa parou. A execução é então desviada para um trecho especial de código, conhecido como **vetor de interrupção**. Esse vetor pode conter instruções diferentes para cada tipo de interrupção que ocorra.

Interrupções são usadas de forma ampla em sistemas microprocessadores e microcontroladores. Em muitos casos, servem para evitar a espera ocupada do processador. Dessa forma, em vez de ter de aguardar a chegada de um pacote pela rede ou fazer a varredura constante do teclado para verificar se uma tecla é acionada, as interrupções podem informar quando esses eventos ocorrem para que providências sejam tomadas. Como a prioridade do tratamento de interrupções é superior às demais, ela só deve realizar tarefas simples e rápidas. Normalmente, os dados são copiados para buffers para que sejam tratados posteriormente.

Os sistemas operacionais multitarefa preemptivos se utilizam do mecanismo de interrupção para chavear o controle do processador entre os diversos processos. Ao passar o controle para uma tarefa, o sistema programa um temporizador para gerar uma interrupção após um tempo determinado. Quando a interrupção ocorre, o sistema salva o contexto da tarefa em execução, recupera o contexto da próxima tarefa a ser executada,

passa o controle para essa tarefa e já programa a próxima interrupção.

Nos sistemas monolíticos, as tarefas de tempo real podem ser disparadas pelo mecanismo de interrupção.

As **interrupções no ESP8266** têm algumas restrições: não podem levar mais que 50 us para executar e têm um esquema de prioridades no qual interrupções de prioridade mais alta podem interromper aquelas de prioridade mais baixa.

Para o ESP8266, uma tarefa é uma unidade de execução normal sem prioridade de interrupção. Ela pode ser interrompida pelas interrupções, mas não pode ser preemptada. O guia de referência da Espressif indica que nenhuma tarefa deveria executar por mais que 15 ms antes de retornar o controle para o SDK, que atua como um sistema operacional, nesse caso. O SDK consegue manter 32 tarefas pendentes com prioridades de 0 a 31 sendo que a tarefa de maior prioridade na fila deve ser a próxima a ser executada.

CAPÍTULO 6

Internet das Coisas e a nuvem

Computação em nuvem é essencial para o funcionamento efetivo e de baixo custo dos dispositivos de IoT. A "nuvem" garante a confiabilidade dos sistemas a baixo custo. Os dispositivos podem permanecer em ambientes hostis e de baixa confiabilidade, desde que seus dados sejam enviados periodicamente para a nuvem. Esse modelo é econômico, pois dispensa a presença de servidores e possibilita a utilização de canais de comunicação de baixa confiabilidade.

6.1 Computação em nuvem (Cloud Computing)

Um termo que tem feito sucesso nos últimos tempos na computação é *Cloud Computing*, ou computação na nuvem. É difícil dizer onde começa a nuvem e até onde vai, visto que boa parte dos seus conceitos já é usada desde os primórdios da internet. Basicamente, podemos dizer que estamos usando a "nuvem" se os serviços essenciais da rede, sejam armazenamento, processamento ou acesso, são hospedados em servidores externos.

O conceito de servidor externo deve considerar o cenário inicial da internet, no qual todas as empresas que queriam estar presentes na internet precisavam instalar seus próprios servidores, fossem de armazenamento, bancos de dados, web ou email. Isso exigia um bom parque computacional das empresas, procedimentos de backup constantes, além de equipamentos para suprir falhas de energia. Com o tempo, esses serviços começaram a ser transferidos para datacenters externos. Em síntese, podemos dizer, então, que computação na nuvem é, basicamente, terceirizar os servidores.

Os primeiros serviços a ser terceirizados foram web e email. Várias empresas iniciaram a oferta de terceirização desses serviços há vários anos, incluindo Google e Microsoft. Com o tempo, surgiram também provedores de aplicações e bancos de dados web. A terceirização desses serviços visa

obter uma alta disponibilidade dos mesmos a baixo custo. São serviços essenciais para as empresas que ofertam serviços na internet, e é muito caro para uma empresa manter servidores de alta disponibilidade conectados à internet.

O conceito de nuvem se tornou mais comum quando os serviços de armazenamento e processamento começaram a se tornar populares na rede. Os serviços de armazenamento e compartilhamento de arquivos, como Dropbox, Microsoft Onedrive ou Google Drive, rapidamente tornaram-se populares, disponibilizando planos básicos gratuitos. E a comunidade começou a salvar seus arquivos na nuvem como opção de backup, acesso remoto e compartilhamento.

A opção de processamento na nuvem foi inicialmente disponibilizada pela Amazon. O serviço conta com a opção de criar máquinas virtualizadas, migrar entre servidores de virtualização, garantindo eficiência e confiabilidade às aplicações. A virtualização em nuvem abriu espaço para as empresas terceirizarem todo o seu parque computacional, desde servidores de bancos de dados até aplicações bancárias, de automação comercial ou industrial. Não apenas aplicações web, mas qualquer sistema, desenvolvido em qualquer linguagem.

Para aplicações de Internet das Coisas, o conceito de nuvem se tornou um complemento importante. Os dispositivos IoT normalmente têm poder de processamento limitado, baixa confiabilidade e alta taxa de erros. Aplicações críticas não devem ficar hospedadas em dispositivos IoT. Esses dispositivos devem enviar suas informações para um elemento centralizador, com maior poder de processamento e disponibilidade. Se esse elemento estiver na nuvem, as aplicações ganham mais flexibilidade e escalabilidade. Além disso, evitam a necessidade de um servidor local de alta disponibilidade.

Aplicações que acessam dispositivos IoT podem ter várias interfaces para a nuvem: web, bancos de dados ou aplicações específicas de automação industrial, comercial ou residencial. Podem incluir aprendizagem de máquina e algoritmos de alta demanda computacional.

6.2 Acesso a aplicações web

Aplicações web foram as primeiras a ir para a nuvem. Apresentam um modelo distribuído, podem ser programadas em diversas linguagens populares e se hospedar em uma infinidade de servidores. Dispositivos IoT podem se conectar à web para enviar e receber informações e comandos ou podem, ainda, disponibilizar suas informações em servidores web, dando origem ao termo Web das Coisas, ou Web of Things (WoT).

O assunto Web das Coisas tem uma comunidade de desenvolvedores, pesquisadores e projetistas que trabalham no futuro da web. Novos padrões estão sendo desenvolvidos, de forma aberta, flexível e escalável, que vão possibilitar a interligação de todos os dispositivos IoT à web para construir aplicações web mais fáceis de usar e integrar. Diversos métodos podem ser usados para acessar os dados dos dispositivos IoT ou enviar comandos, desde os métodos tradicionais até os serviços web conhecidos como *Web Services*.

Os principais métodos de acesso web são POST e GET. Esses métodos são acionados no cliente web para acessar dados no servidor. Ambos possibilitam o envio de parâmetros e recebem, como resposta, uma página em HTML, que pode conter os dados de resposta à solicitação. Uma página web é normalmente criada com um formulário para chamar os métodos POST ou GET, mas a chamada pode ser automatizada em programação para IoT.

6.3 Web Services

Web Services é uma proposta para integração e comunicação entre diferentes sistemas usando a plataforma web. Essa solução possibilita que aplicações desenvolvidas em diferentes tecnologias possam interagir mesmo que sejam escritas em diferentes linguagens e executem em diferentes plataformas. As funções e os dados são definidos e enviados em formato XML. Formatos intermediários, como JSON, CSV ou XML, são usados para que as aplicações disponibilizem e acessem os métodos remotos. E a comunicação é toda encapsulada em requisições HTTP, o que facilita seu acesso e o controle em firewalls.

A proposta segue tendências já utilizadas em diversas outras tecnologias como RPC, Activex, Corba ou Java RMI, que possibilitam que aplicações

distribuídas se comuniquem entrem si, trocando informações, chamando métodos ou objetos remotos. Web Services têm o diferencial de se utilizar da web para realizar o acesso aos métodos e dados remotos. Utilizam, assim, os mesmos princípios e servidores web, incluindo comunicação em modo texto e acesso cliente/servidor.

O cenário de Internet das Coisas é extremamente propício a usar Web Services para integrar o cenário, essencialmente distribuído, em uma ferramenta web capaz de gerenciar todas as "coisas" e disponibilizar seus serviços para as diversas interfaces. O inconveniente para essa integração é que as bibliotecas de integração Web Services exigem, normalmente, alta disponibilidade de memória para manipular as estruturas de dados, o que normalmente não está disponível em dispositivos IoT de baixo custo. Porém existem algumas implementações que podem ser usadas, com restrições.

A plataforma Arduino conta com uma biblioteca para Web Services denominada **aJson**, uma implementação do JSON (JavaScript Object Notation), formato de dados para Web Services criado a partir de um subconjunto da linguagem JavaScript. Essa plataforma possibilita a interpretação de dados usando essa tecnologia, mas está restrita aos módulos Arduino com maior poder de processamento e memória. Os módulos mais simples, baseados no microcontrolador Atmel ATmega168, não suportam a biblioteca aJson.

O módulo NodeMCU, baseado no microcontrolador Espressif ESP8266, também tem uma biblioteca de implementação de JSON. É uma variação do JSON, chamada cJson, que tem a mesma finalidade. O SDK da Espressif também conta com biblioteca JSON que é usada, entre vários exemplos, para conectar à plataforma de nuvem IBM Watson.

6.4 Bancos de dados

SGBD (Sistemas de Gerenciamento de Bancos de Dados) são softwares especiais que gerenciam o armazenamento de dados. Os dados são normalmente organizados em tabelas, as quais mantêm relacionamentos de acordo com a lógica de organização dos dados. O projeto de bancos de dados envolve várias fases: modelagem entidade-relacionamento, álgebra

relacional e descrição em linguagem SQL.

A modelagem **entidade-relacionamento** é o primeiro passo para o projeto de bancos de dados relacionais. Essa fase permite identificar quais informações devem ser armazenadas e como elas se relacionam, como é sua cardinalidade e quais informações caracterizam unicamente um elemento em um conjunto de dados. Essa etapa gera como resultado um diagrama chamado entidade-relacionamento, que indica a estrutura geral de organização dos dados.

A segunda etapa do projeto de banco de dados é construir a lógica de organização e localização de dados. Chamamos de álgebra relacional, devido aos operadores e à representação matemática utilizada. Essa fase facilita a representação dos comandos em linguagem SQL, que seria a próxima etapa. Nem sempre essa fase é desenvolvida com o formalismo da álgebra relacional. Muitos desenvolvedores já organizam a lógica relacional usando a linguagem SQL.

A linguagem SQL (Structured Query Language – Linguagem de Consulta Estruturada) é uma linguagem usada para acessar os sistemas de gerenciamento de bancos de dados. É uma linguagem universal, usada em quase todos os bancos de dados, apesar de algumas pequenas variações entre fabricantes. Conta com subconjuntos da linguagem para funções específicas, como a DML (Linguagem de Manipulação de Dados) e a DDL (Linguagem de Definição de Dados). A DML contém a parte da linguagem que possibilita localizar, inserir ou atualizar os dados. A DDL contém a parte da linguagem que define a estrutura dos dados e das tabelas.

Vários SGBDs estão disponíveis, alguns de forma livre e gratuita, como o MySQL e o Postgres, outros pagos, como o Microsoft SQL Server e o Oracle Database. Os SGBDs tendem a ser instalados sempre na nuvem, visto que sua instalação em servidores particulares exige computadores de alto desempenho, além de backups e atualizações constantes. Vários provedores de serviços na internet disponibilizam acesso a SGBDs. Alguns fornecem até cotas de acesso gratuitas para usuários de baixa utilização.

Para que uma aplicação acesse um SGBD, é necessário que a linguagem e a plataforma suportem o acesso ao referido SGBD. Em dispositivos IoT isso nem sempre é possível. A plataforma Arduino contém bibliotecas para

acesso ao MySQL, um SGBD gratuito e disponível em diversos provedores. Isso é uma grande vantagem dessa plataforma, que pode ser usada também para programar o ESP8266. Usando a biblioteca disponível, é possível enviar comandos diretamente a um servidor MySQL.

6.5 MQTT

Uma solução bastante atraente para IoT, que surgiu recentemente, foi o protocolo MQTT. Ele usa o conceito de broker, que basicamente é um software servidor que recebe solicitações e as repassa, quando solicitado. O broker atua como servidor nos dois sentidos, tanto recebendo quanto enviando os dados. Assim, evita a necessidade de manter um servidor na rede, atuando sempre nessa função.

Um broker atuaria da mesma forma que um servidor web ou SGBD, com a diferença que tem um modelo mais simplificado e orientado a aplicações que repassam dados simples. A sintaxe é mais simples, e a interface, mais leve. A maioria dos sistemas IoT suporta acesso ao protocolo MQTT, incluindo o Arduino e o nodeMCU.

A figura 6.1 apresenta o funcionamento do MQTT. Os dispositivos IoT, à esquerda, geram dados por meio de sua capacidade de sensoriamento ou de interfaceamento com os diversos tipos de dispositivos. Esses dados são registrados em um broker e ficam disponíveis para ser acessados por dispositivos de interface; para apresentá-los ou utilizá-los em aplicações, podem ser acessados por outros dispositivos IoT. Um dispositivo responsável por um acionamento, por exemplo, pode ler em um broker se deve manter o acionamento ligado, desligado ou até mesmo ler um parâmetro de configuração para o dispositivo.

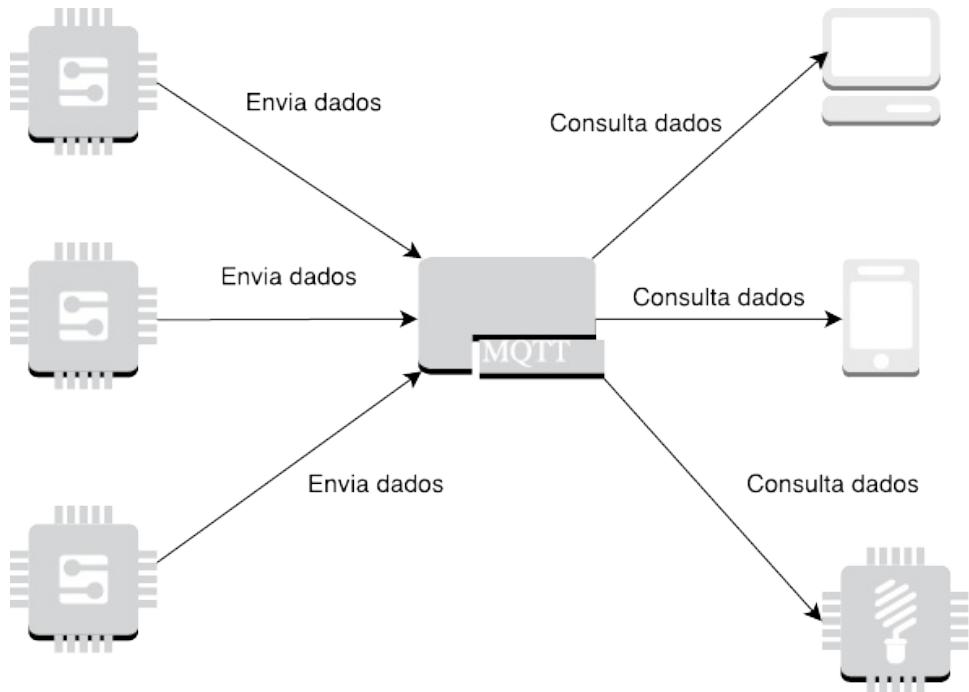



Figura 6.1 – Funcionamento básico do MQTT.

6.6 Plataformas comerciais de nuvem

A perspectiva de criar máquinas virtuais, instalar serviços diversos incluindo bancos de dados, web, DNS e outros, de forma exclusiva e isolada, só é possível nos serviços pagos de nuvem. Há várias plataformas de nuvem disponíveis. Algumas delas já têm interfaces para IoT que possibilitam novas abordagens e novos serviços. Os principais são: Microsoft Azure, Google Cloud Plataform, IBM Watson IoT Plataform, Amazon Web Services IoT. A maioria deles oferece um nível básico de serviço gratuito, possivelmente exigindo a inserção dos dados de cartão de crédito e alertando que haverá cobranças caso excedam-se os níveis de serviços gratuitos.

6.6.1 Microsoft Azure

A plataforma de nuvem da Microsoft, conhecida como Azure, oferece vários serviços interessantes para aplicações IoT, desde virtualização de máquinas remotas, incluindo protocolos de comunicação de dados com IoT, como MQTT e Web Services, bancos de dados, algoritmos de análise de *big data* e aprendizagem de máquina.

A inscrição no serviço é gratuita e inclui 750 dólares de crédito para ser usado durante um mês. Após esse período, haverá cobrança no cartão de crédito cadastrado, bem como se ultrapassar o valor de 750 dólares no primeiro mês. Por isso, se o interesse for apenas testar a ferramenta, fique atento para as possibilidade de cobrança. Está prevista uma licença universitária para as universidades que aderirem ao DreamSpark, programa acadêmico da Microsoft.

A principal interface IoT do Microsoft Azure é **Hub IoT**. Funciona como um receptor de mensagens dos dispositivos IoT. Pode trabalhar em diversos protocolos como HTTP, AMQP e MQTT. Engloba bibliotecas de integração para diversas plataformas populares IoT, incluindo Raspberry Pi, Adafruit Feather e Intel Edison. Para o microcontrolador ESP8266, há kits baseados no SparkFun Thingdev e Adafruit Huzzah Feather, duas placas comerciais que utilizam esse microcontrolador. Ambas são baseadas no Arduino, o que estende sua utilização às demais placas baseadas nesse microcontrolador.

6.6.2 Amazon AWS

A Amazon foi a pioneira na oferta de serviços na nuvem. Iniciou em 2006, com serviços específicos de armazenamento, e depois incluiu o serviço de processamento chamado EC2 (Elastic Cloud Computing) – baseado no virtualizador Xen. Detém a maior fatia do mercado, proporcionando confiabilidade, ampla oferta de serviços e experiência.

O suporte à Internet das Coisas abrange uma plataforma específica na Amazon, AWS IoT, que disponibiliza doze meses gratuitos para integração e testes. Disponibiliza também um broker MQTT e possibilita a integração a todos os outros serviços em nuvem, incluindo acesso a bancos de dados, processamento e bibliotecas de inteligência artificial e aprendizagem de máquina.

6.6.3 IBM Watson

A IBM também tem uma infraestrutura de nuvem para dar suporte às diversas aplicações, conhecida como IBM Watson. Além da virtualização típica das aplicações em nuvem, conta com uma interface robusta de

Internet das Coisas. Totalmente gerenciável, tem também acesso MQTT e um ambiente de desenvolvimento que facilita a integração e a criação de aplicações.

A IBM oferece uma opção gratuita para integração de dispositivos IoT na plataforma IBM Watson. Com capacidade de 500 dispositivos, 500 acessos para aplicações, 200 MB de dados trocados e analisados, o plano gratuito é capaz de produzir provas de conceito e facilitar o aprendizado de integração de dispositivos IoT à nuvem.

6.6.4 Google Clould Platform

Apesar de ser pioneiro em serviços em nuvem, como o Gmail, o Google não atua de forma intensiva na nuvem. Começou a dispor seus serviços em 2008, mas apenas em 2013 deu início à operação do serviço Google Compute Engine, para o processamento de máquinas virtuais. Não tem muitos parceiros, e sua participação no mercado é pequena.

O Google sempre apostou no uso de suas ferramentas, linguagens e plataformas para oferecer serviços na nuvem. Começou assim com o serviço Google Apps, hoje denominado G Suite, que pode ser considerado o primeiro serviço na nuvem personalizado para o cliente. Por esse serviço, o Google disponibiliza servidores de email, baseados no Gmail, servidores web e registro de domínio, com aplicações e persistência, utilizando, inicialmente, a linguagem Python, e recentemente com uma linguagem de script denominada Apps Script, similar à JavaScript. Essa plataforma, atualmente, está mais voltada para as ferramentas de escritório Google Docs.

Para as aplicações típicas de IoT, a melhor opção de utilização de nuvem é o protocolo MQTT. Leve, disponível de forma gratuita por dezenas de provedores e totalmente aberto. Suas limitações estão na ausência de persistência, ou seja, suas informações não são salvas, bem como na falta de poder de processamento, visto que é exclusivo para a troca de mensagens.

Aplicações que demandam armazenamento e processamento remoto devem considerar uma das plataformas comerciais de nuvem, realizando a hospedagem dos serviços de bancos de dados e web, com acesso pelo

protocolo MQTT ou, ainda por acesso web. A oferta de serviços na nuvem cresce intensamente a cada dia, com soluções específicas para IoT, incluindo algoritmos, protocolos e serviços. Vale a pena pesquisar o que cada provedor oferece antes de contratar seus serviços.

PARTE II

Implementação

CAPÍTULO 7

Programação NodeMCU com a linguagem Lua

Lua é uma linguagem interpretada de alto nível que possibilita uma programação simples e rápida para os módulos com o microcontrolador ESP8266. O ambiente de desenvolvimento NodeMCU tem uma interface amigável e proporciona resultados rápidos para o desenvolvimento de aplicações IoT com um conjunto de sensores e periféricos suportados.

O primeiro ambiente de programação a ser apresentado para o microcontrolador ESP8266 é baseado na linguagem de programação Lua. Desenvolvida no Departamento de Ciência da Computação da PUC-Rio, e amplamente utilizada em todo o mundo, seja para aplicações embarcadas, gráficas ou animações, a linguagem Lua é poderosa, mas simples, leve e aberta, facilitando sua utilização e integração nas mais diversas aplicações.

A integração da linguagem Lua com o microcontrolador ESP8266 gerou um novo nome: NodeMCU, disponível no site *http://nodemcu.com*, com vários exemplos e fóruns. O nome se popularizou de tal forma que é mais fácil localizar informações a respeito desse microcontrolador, bem como encontrá-lo nos sites de fornecedores pelo nome NodeMCU do que pelo nome ESP8266.

7.1 Gravando o firmware

O primeiro passo para desenvolver uma aplicação em Lua para o NodeMCU é gerar o firmware com as bibliotecas necessárias. A geração do firmware é feita de forma gratuita por meio do site *http://nodemcu-build.com*. Após escolher as bibliotecas necessárias, sendo as mais comuns já sugeridas, o firmware é gerado e enviado por email. Depois desse processo, o firmware pode ser gravado diretamente no módulo pela porta USB, usando o programa NodeMCU-flasher para o sistema operacional Windows, disponível em *https://github.com/nodemcu/nodemcu-flasher,* ou o

software Esptool, disponível em *https://github.com/themadinventor/esptool,* escrito em Python 2, que pode ser executado tanto no sistema operacional Linux quanto no Windows. A figura 7.1 mostra o software NodeMCU-flasher. É importante conferir a porta "COM Port" na qual está ligada o NodeMCU. Essa porta é atribuída automaticamente pelo Windows quando o dispositivo é conectado a uma porta USB e pode ser visualizada pelo Gerenciador de Dispositivos do Windows.

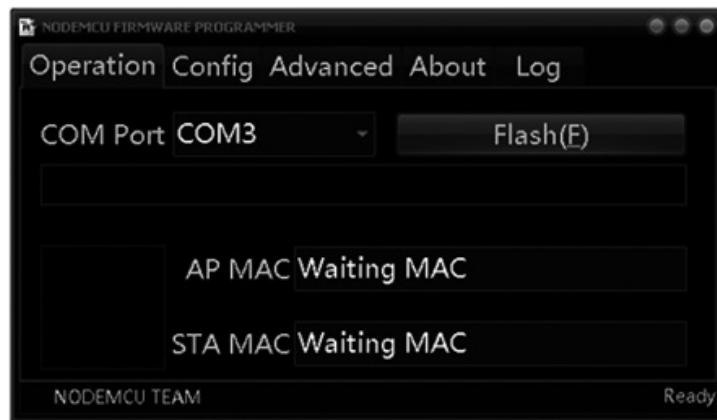



Figura 7.1 – Software para gravação do firmware Lua.

A figura 7.2 mostra o Gerenciador de Dispositivos do Windows, indicando a porta em uso para o NodeMCU. A descrição sempre indicará "USB-SERIAL CH340", e a referência à porta estará entre parênteses.

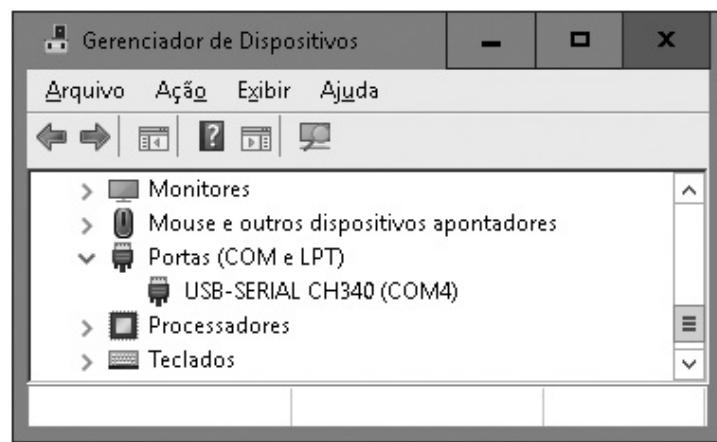



Figura 7.2 – Porta serial do NodeMCU no Gerenciador de Dispositivos.

Caso o NodeMCU já tenha um sistema instalado e o processo de gravação de firmware seja de atualização, ou exista outro sistema gravado, é preciso executar um procedimento para atualizar o bloco de dados de inicialização antes de gravar o firmware. Para fazer isso no NodeMCU-flasher, basta

gravar usando a configuração “INTERNAL://DEFAULT” e “INTERNAL://BLANK”, nos endereços 0x7C00 e 0x7E00, respectivamente, e em letras maiúsculas, como mostrado na figura 7.3.

Figura 7.3 – Tela de configuração para atualização do firmware do NodeMCU.

Após esse procedimento, basta gravar o firmware gerado clicando na engrenagem ao lado e escolhendo o arquivo do firmware. Em seguida, o NodeMCU reinicia e ele já estará pronto.

Após gravar o firmware, o NodeMCU já responde a comandos da linguagem Lua de forma interpretada. Por ser uma linguagem interpretada, Lua dispensa a fase de compilação. Para acessar os comandos Lua, é possível usar um terminal de acesso à interface serial; sugiro Putty ou HyperTerminal, considerando o acesso pelo Windows. A velocidade deve ser configurada para 115.200 baud.

O código a seguir mostra a resposta obtida por meio do acesso à porta serial na qual está ligado o NodeMCU com o programa Putty. O comando `node.restart()` foi executado. Observam-se as informações referentes ao firmware gerado, incluindo as bibliotecas adicionadas: `adc`, `dht`, `enduser_setup`, `file`, `gpio`, `http`, `i2c`, `mqtt`, `net`, `node`, `pwm`, `tmr`, `uart`, `wifi`.

```
ets Jan 8 2013,rst cause:2, boot mode:(3,6)

load 0x40100000, len 26404, room 16
tail 4
chksum 0xa1
```

```
load 0x3ffe8000, len 2212, room 4
tail 0
chksum 0x71
load 0x3ffe88a4, len 136, room 8
tail 0
chksum 0xd5
csum 0xd5
▒▒▒'▒{▒▒g|▒dlll`▒▒{▒l▒l
                        ▒▒

NodeMCU custom build by frightanic.com
        branch: master
        commit: 22e1adc4b06c931797539b986c85e229e5942a5f
        SSL: false
        modules:
adc,dht,enduser_setup,file,gpio,http,i2c,mqtt,net,node,pwm,tmr,
                uart,wifi
build built on: 2017-04-03 20:00
powered by Lua 5.1.4 on SDK 2.0.0(656edbf)
lua: cannot open init.lua
```

7.2 Ambiente de desenvolvimento

Uma ferramenta de programação Lua para o NodeMCU conhecida por ESPlorer está disponível em *http://esp8266.ru/esplorer* e funciona tanto para Windows quanto para Linux, visto que foi desenvolvida em Java. A figura 7.4 apresenta a interface do ESPlorer. Ele possibilita programar e acompanhar a comunicação com o NodeMCU em tempo real.

Figura 7.4 – Software ESPlorer para programação Lua no NodeMCU.

O site principal do NodeMCU traz alguns exemplos de aplicações interessantes. Entre eles, métodos rápidos para acessar a rede WiFi, alterar os valores das saídas de I/O, ler as entradas de I/O, cliente e servidor HTTP, PWM, temporizadores, servidor telnet e interface com um sensor de temperatura.

7.3 Acesso à rede WiFi

O código para acesso à rede WiFi é bem simples, como quase tudo na linguagem Lua. Veja a seguir.

```
wifi.setmode(wifi.STATION)
print(wifi.sta.getip())
--nil ou o endereço se já tiver conectado
wifi.setmode(wifi.STATION)
wifi.sta.config("ssid","senha")
wifi.sta.connect()
while (wifi.sta.getip() == nil) do
    print("Conectando...")
    tmr.delay(1000000)
end
print("Conectado, IP é "..wifi.sta.getip())
```

Para modo AP:

```
cfg={ssid="teste", pwd="senha123"}
wifi.setmode(wifi.SOFTAP)
wifi.ap.config(cfg)
```

```
wifi.ap.dhcp.start()
```

Os comandos fazem parte da biblioteca `wifi`, que deve ser referenciada sempre. O comando `wifi.sta.getip()` retorna o endereço IP, caso esteja conectado. O comando `wifi.setmode()` define o modo de operação, que pode ser `wifi.STATION`, para funcionar como uma estação da rede; `wifi.SOFTAP`, para atuar apenas como Ponto de Acesso (AP); `wifi.STATIONAP`, qua habilita ambos os modos, de forma simultânea, possibilitando o uso como repetidor; e `wifi.NULLMODE`, que desliga o WiFi.

A função `wifi.sta.config()` só funciona se o modo `STATION` estiver habilitado. Ele possibilita informar o SSID e a senha, para posterior conexão com o método `wifi.sta.connect()`. O método `wifi.sta.connect()` não é instantâneo e não fica em espera ocupada. Caso seja necessário aguardar a conexão, esta pode ser confirmada verificando-se a função `wifi.sta.getip()`, que retornará `nil` se a conexão não tiver sido feita. Um temporizador de 1 s foi inserido para garantir que o processador não fique travado nesse procedimento.

Vale notar que o ESP8266 salva as configurações de rede WiFi já conectadas. Assim, pode ser que a conexão para uma rede conhecida ocorra mesmo antes de o método `wifi.sta.connect()` ser chamado. O mesmo vale quando compilado e executado em outros ambientes, como o Arduino. As redes são mantidas mesmo quando o firmware é recarregado.

7.4 Acessando GPIO

O seguinte exemplo faz acesso a pinos de I/O, GPIO:

```
pinw=0
gpio.mode(pinw,gpio.OUTPUT)
gpio.write(pinw,gpio.HIGH)
pinr=1
gpio.mode(pinr,gpio.INPUT)
print(gpio.read(pinr))
```

Os modos de acesso, configurados em `gpio.mode()`, são `gpio.OUTPUT` ou `gpio.INPUT`. A leitura usa o método `gpio.read()` e a escrita `gpio.write()`. Os valores para ligado e desligado são, respectivamente, `gpio.HIGH` e `gpio.LOW`.

7.5 HTTP Client

Um método simples para acessar uma página em servidor web pode ser visto no próximo exemplo:

```
wifi.setmode(wifi.STATION)
print(wifi.sta.getip())
--nil ou IP previamente configurado
wifi.sta.config("ssid","password")
wifi.sta.connect()

while (wifi.sta.getip() == nil) do
    print("Conectando...")
    tmr.delay(1000000)
end

print("Conectado, IP é "..wifi.sta.getip())
http.get("http://httpbin.org/ip", nil, function(code, data)
    if (code < 0) then
        print("Falha na requisição HTTP")
    else
        print(code, data)
    end
end)
```

Esse exemplo usa a biblioteca `http`, que implementa métodos para acesso do NodeMCU como cliente. Nesse caso, apenas o método `http.get()` é usado. Esse método tem três parâmetros: a *url* a ser acessada, cabeçalhos opcionais e uma função chamada callback, ou seja, uma função que é chamada como retorno quando a reposta é recebida. Essa característica da linguagem Lua é muito interessante para esse tipo de sistema distribuído, cuja resposta depende de outro sistema remoto.

Caso o NodeMCU esteja conectado à rede WiFi com acesso à Internet, a resposta para essa execução deve ser algo do tipo:

```
200 {
    "origin": "189.12.105.144"
}
```

7.6 Servidor HTTP

O próximo exemplo mostra como o NodeMCU pode atuar como um servidor web. Nesse caso, ele escuta solicitações recebidas pela rede, na porta 80, respondendo ao receber as solicitações:

```
-- um servidor http simples
wifi.setmode(wifi.STATION)
print(wifi.sta.getip())
--nil ou IP previamente configurado
wifi.sta.config("ssid","senha")
wifi.sta.connect()

while (wifi.sta.getip() == nil) do
    print("Conectando...")
    tmr.delay(1000000)
end

print("Conectado, IP é "..wifi.sta.getip())
srv = net.createServer(net.TCP)
srv:listen(80, function(conn)
    conn:on("receive", function(sck, payload)/
        print(payload)
        sck:send("HTTP/1.0 200 OK\r\nContent-Type: text/html\r\n\r\n
            <h1> Olá, mundo! </h1>")
    end)
    conn:on("sent", function(sck) sck:close() end)
end)
```

Esse exemplo não usa a biblioteca `http`, que é exclusiva para acesso do NodeMCU como cliente web. A biblioteca usada é a biblioteca básica de rede `net`, que tem funções para manipulações de soquetes TCP. A função `net.createServer()` recebe como parâmetro o protocolo a ser usado, TCP, nesse caso, e retorna uma variável que controla a conexão. A essa variável será atribuída a função `listen()`, indicando a porta usada (80) para receber as solicitações, e uma função callback, que será chamada sempre que uma solicitação da rede nessa porta for recebida. A função imprime a solicitação recebida para a saída serial, que será apresentada no ESPlorer, e retorna um conteúdo em linguagem HTML com os dizeres “Ola, mundo!”. Ao acessar o endereço IP do NodeMCU em um navegador web, essa mensagem deve ser exibida, como pode ser visto na figura 7.5.

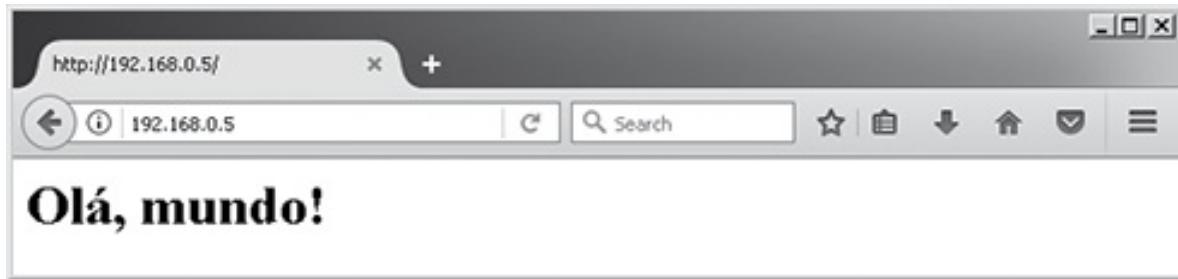



Figura 7.5 – Resultado da execução do servidor web em Lua.

Cada vez que o acesso for feito, uma mensagem será exibida pelo NodeMCU, como pode ser visto a seguir:

```
GET /favicon.ico HTTP/1.1
Host: 192.168.100.12
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36
Accept: */*
Referer: http://192.168.100.12/
Accept-Encoding: gzip, deflate, sdch
Accept-Language: pt-BR,pt;q=0.8,en-US;q=0.6,en;q=0.4,zh-
CN;q=0.2,zh;q=0.2,fr;q=0.2
```

Essa mensagem exibe informações sobre a solicitação recebida.

7.7 Temporizadores

Apesar de não ter um sistema multitarefa, o NodeMCU contém sete temporizadores que podem ser usados para automatizar tarefas de forma paralela. Segue o exemplo:

```
tmr.alarm(1,5000,tmr.ALARM_AUTO,function() print("alarme 1") end)
tmr.alarm(0,1000,tmr.ALARM_AUTO,function() print("alarme 0") end)
tmr.alarm(2,2000,tmr.ALARM_AUTO,function() print("alarme 2") end)
```

Nesse exemplo, três temporizadores são usados pra criar alarmes. A função `tmr.alarm()` associa o temporizador a uma função ou a um trecho de código. Recebe como parâmetro o número do temporizador, de 0 a 6, o tempo de intervalo, em milissegundos; o modo do temporizador, que pode ser `tmr.ALARM_SINGLE`, para executar apenas uma vez; `tmr.ALARM_SEMI`, que

requer o acionamento manual pela função `tmr.start()`; ou `tmr.ALARM_AUTO`, que já inicia automaticamente após sua definição.

Os temporizadores são muito úteis para tratar eventos de rede, ler e tratar dados de sensores, ou programar saídas que têm comportamento periódico, como um LED piscando.

O próximo exemplo mostra como os temporizadores podem ser usados para paralelizar eventos. Ele usa um servidor web para comtrolar um LED, que pode permanecer em três estados: ligado continuamente, piscando ou desligado.

```
-- Conexão na rede Wifi
wifi.setmode(wifi.STATION)
wifi.sta.config("ssid","senha")
wifi.sta.connect()
tmr.alarm(1, 1000, tmr.ALARM_AUTO, function()
    if wifi.sta.getip() == nil then
        print("Conectando...")
    else
        tmr.stop(1)
        print("Conectado! IP é "..wifi.sta.getip())
    end
end)

pin=0
led=false
-- Função para piscar o LED
function pisca()
    if led then
        led=false
        gpio.write(pin,gpio.LOW)
    else
        led=true
        gpio.write(pin,gpio.HIGH)
    end
end

-- Temporizador
tmr.alarm(0,1000, tmr.ALARM_AUTO, function() pisca() end )
```

```
-- Servidor web
srv=net.createServer(net.TCP)
srv:listen(80,function(conn)
    conn:on("receive", function(client,request)
        page = "<h1>Exemplo web - Pisca</h1>"
        page = page.."<p><a href=\"?com=PISCA\"><button><b>PISCA
LED</b>
            </button></a>"
        page = page.."<p><a href=\"?com=DESLIGA\"><button>
<b>DESLIGA</b>
            </button></a>"
        client:send(page)
        client:close()
        if(string.find(request, "PISCA")) then
            tmr.start(0)
        elseif(string.find(request, "DESLIGA")) then
            tmr.stop(0)
        end
    end)
    conn:on("sent", function(client) client:close()
    end)
end)
```

O exemplo mostra a aplicação de dois temporizadores. O primeiro é usado para repetir a expressão "Conectando...", até que a rede WiFi esteja disponível. O mesmo havia sido feito usando o laço `while` em exemplo anterior. O segundo temporizador tem a função de fazer o LED piscar com a frequência de um segundo. E o seu acionamento é feito por um servidor web.

A terceira parte do código apresenta um servidor web que recebe requisições na porta 80, respondendo com um código em HTML. O código apresenta dois botões, como pode ser visto na figura 7.6. Ao clicar no botão "PISCA LED", a URL é solicitada novamente ao servidor, porém com a inclusão da palavra "PISCA", como atributo para o campo HTML `conn`. Este campo é tratado no retorno da solicitação, que tenta localizar a palavra "PISCA" ou "DESLIGA". Ao clicar no botão "DESLIGA", a URL é também solicitada novamente ao servidor, desta vez com a inclusão da

palavra “DESLIGA”, como atributo para o campo `conn`. O código analisa a presença dessas palavras na solicitação. Caso sejam encontradas, uma ação sobre o temporizador é executada, parando ou iniciando o seu funcionamento.

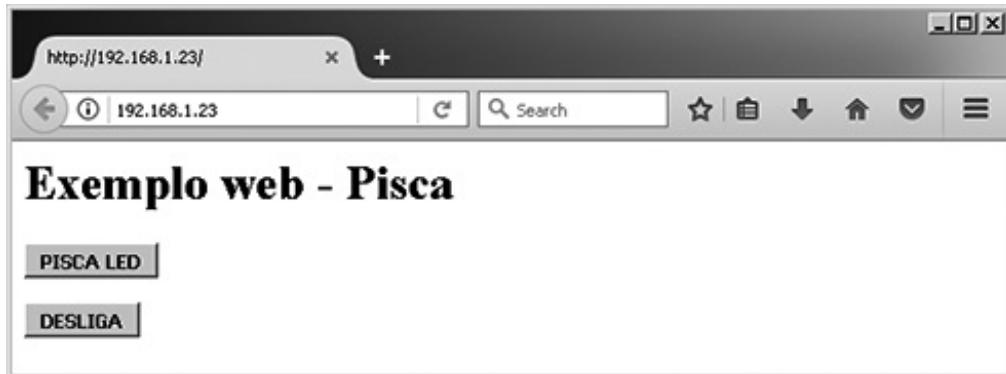



Figura 7.6 – Resultado da execução do programa pisca LED.

7.8 Modo de autoconfiguração

O NodeMCU tem também um modo de autoconfiguração que possibilita distribuir o dispositivo sem nenhuma configuração prévia. Nesse modo, ao ligar o dispositivo, o WiFi entra no modo AP e ele pode ser configurado por qualquer interface web em um computador ou smartphone. A biblioteca `enduser_setup` é necessária para essa configuração e deve ser adicionada ao firmware. O seguinte código do exemplo pode ser usado:

```
enduser_setup.start(
    function()
        print("Conectado ao WiFi como:" .. wifi.sta.getip())
        enduser_setup.stop()
    end,
    function(err, str)
        print("enduser_setup: Erro #" .. err .. ": " .. str)
    end);
```

Nessa configuração, o NodeMCU inicia um AP com o SSID `SetupGadget_XXXXXX`. Basta conectar um dispositivo à essa rede e fazer o acesso web a qualquer endereço, como *http://qualquer.com*, para obter a tela de configuração, como mostra a figura 7.7.

Figura 7.7 – Tela de autoconfiguração do NodeMCU.

A função `enduser_setup.start()` inicia o AP de configuração. A função `enduser_setup.stop()` finaliza essa opção, que não deve ser mantida habilitada por questões de segurança.

É possível alterar, ainda, o SSID inicial usando a função `enduser_setup.manual(true)` e informando o SSID na função de configuração da rede WiFi.

O ESP8266 armazena até cinco redes WiFi, com as respectivas senhas, com as quais ele já tenha se conectado. Assim, caso seja usada a função `enduser_setup.start()`, a rede digitada no navegador será armazenada nessa lista. Graças a essa lista, caso o módulo encontre uma rede disponível, ele vai se conectar a essa rede e sair da função de autoconfiguração.

Por todas essas funcionalidades, o NodeMCU usando a linguagem Lua é bastante interessante para a integração e a programação de dispositivos IoT. A linguagem é simples e compacta, não exige declaração de variáveis e contém um conjunto de bibliotecas capaz de possibilitar a implementação da maioria das aplicações.

O NodeMCU com a linguagem Lua só não será interessante para quem quiser ter um controle maior do hardware, além de mais bibliotecas que ainda não estejam disponíveis em Lua, pois terá de optar pelo SDK RTOS, ou para quem já tem experiência com o ambiente de desenvolvimento Arduino, ou precisa de alguma biblioteca que já esteja disponível para o Arduino, como para acesso ao MySQL. Nesse caso, o ambiente de desenvolvimento Arduino será mais indicado.

A documentação completa do NodeMCU e de suas bibliotecas pode ser encontrada em *https://nodemcu.readthedocs.io*.

CAPÍTULO 8

Programação na IDE Arduino

A IDE Arduino é bastante conhecida dos entusiastas por microcontroladores e microeletrônica. Seu lançamento como hardware aberto e o impulso à indústria eletrônica na China tornaram essa plataforma bastante acessível tanto para amadores e estudantes quanto para empresas que lançaram vários produtos de automação e eletrônica em geral usando essa plataforma.

A mesma IDE Arduino pode ser usada para programar o módulo NodeMCU, ou qualquer módulo que utilize o microcontrolador ESP8266. E a maior parte dos exemplos e programas disponíveis para o Arduino pode também ser compilada e carregada nos módulos com esse microcontrolador. As bibliotecas do Arduino também devem ser compatíveis, favorecendo sua integração a diversos módulos de sensores, atuadores e de comunicação que já estão integrados ao Arduino. É necessário observar, no entanto, as diferenças no hardware. A maior delas diz respeito ao número de portas de entrada analógicas. Enquanto os módulos Arduino normalmente têm quatro entradas analógicas, o ESP8266 tem apenas uma.

Diferente da linguagem Lua, a programação com ambiente Arduino não requer firmware. O código, escrito em um subconjunto da linguagem C++, é compilado em um bloco monolítico e gravado diretamente na memória de programa do NodeMCU. A maior vantagem desse processo é garantir maior poder de processamento, visto que o processador ficará totalmente dedicado ao programa. Como desvantagem, deve ser observado que não há nenhum tipo de gerência de hardware e software disponível. Enquanto a linguagem Lua disponibiliza algumas estruturas, como temporizadores e funções do tipo *callback* para tratar eventos, o programa no ambiente Arduino deve contar com todo o tratamento necessário para seus eventos e

as temporizações. Isso pode levar a mais erros de programação e travamentos.

Para dar início aos trabalhos, é necessário fazer o download do ambiente de desenvolvimento Arduino (*http://www.arduino.cc*). Depois de instalado o ambiente Arduino, é preciso ir em Arquivos e depois Preferências, e a tela da figura 8.1 será apresentada. Em seguida, adicione ao campo **URLs Adicionais de Gerenciadores de Placas** a URL *http://arduino.esp8266.com/stable/package_esp8266com_index.json*.

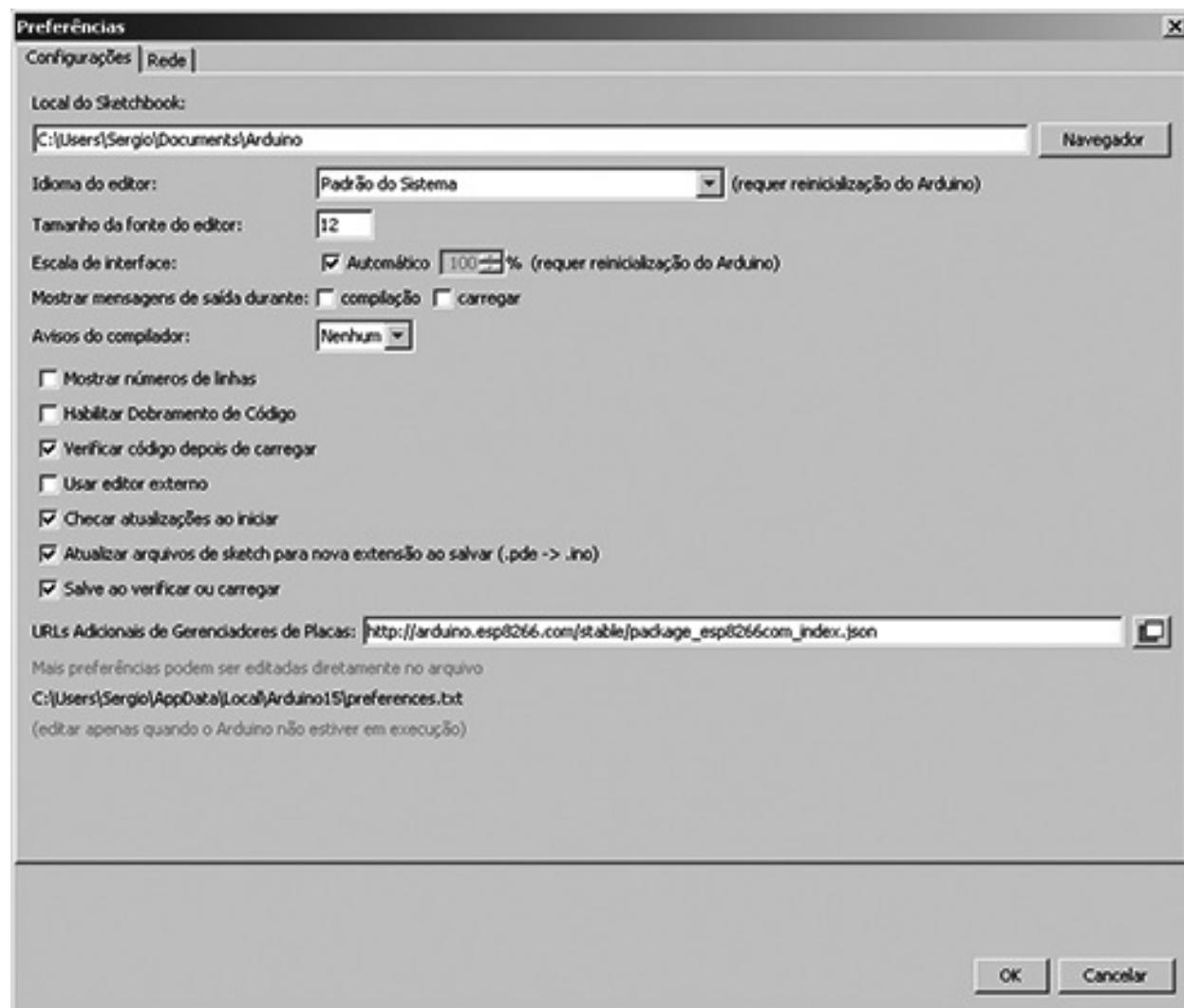



Figura 8.1 – Tela de preferências da IDE Arduino.

Depois, deve-se acessar o menu **Ferramentas** e, então, a **Placa** e o **Gerenciador de Placas**. Na caixa de diálogo que vai aparecer, mostrada na figura 8.2, há uma opção para instalar o suporte à placa ESP8266. Deve ser instalada a versão mais nova disponível.

Depois da configuração, é possível compilar e carregar os programas normalmente no módulo NodeMCU, bem como em qualquer outro módulo baseado no ESP8266.

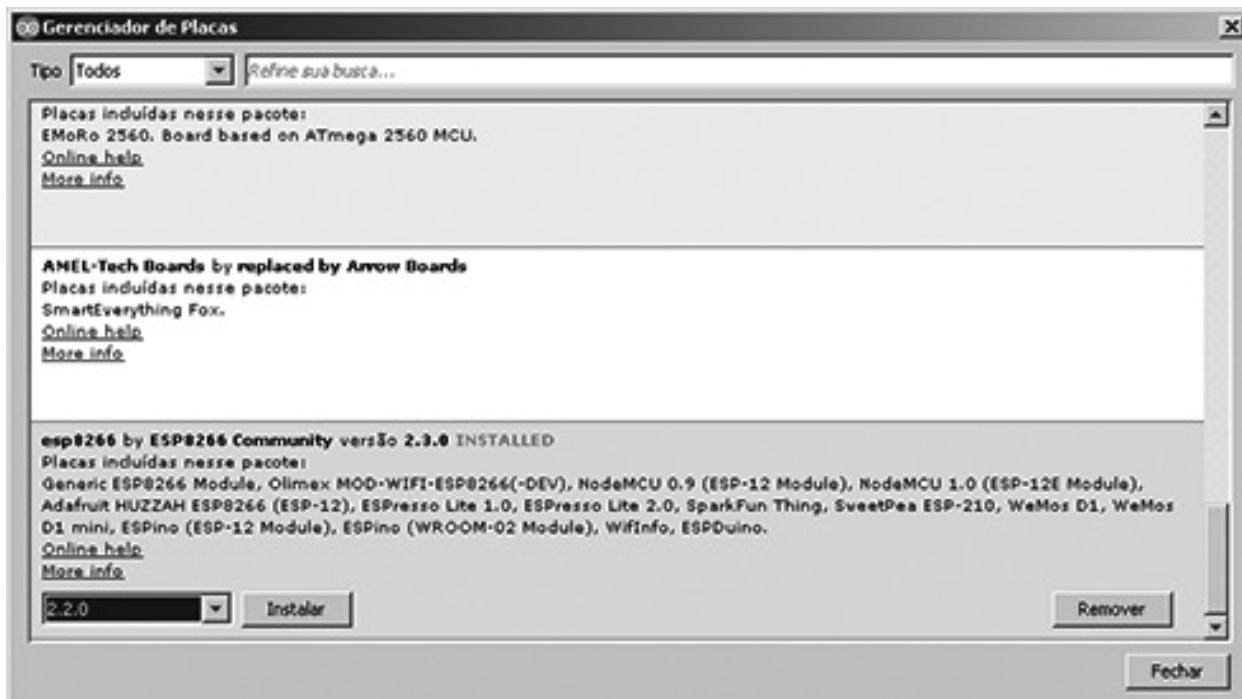



Figura 8.2 – Tela para a instalação do suporte ao ESP8266 na IDE Arduino.

8.1 Exemplos no ambiente Arduino

8.1.1 Pisca LED

O primeiro exemplo é o clássico pisca LED. Será usado o próprio LED da placa, que está na porta GPIO 16, correspondente ao pino `D0`.

```
#include <ESP8266WiFi.h>

void setup() {
    pinMode(D0, OUTPUT);
}

void loop() {
    digitalWrite(D0, LOW);
    delay(1000);
    digitalWrite(D0, HIGH);
    delay(1000);
```

```
}
```

O ambiente Arduino conta sempre com duas funções importantes: `setup()`, executada sempre ao iniciar ou reiniciar o microcontrolador; e a função `loop()`, continuamente executada, em laço sem fim. Na função `setup()`, são alocados os procedimentos relativos a configurações e inicializações. Na função `loop()`, os procedimentos que devem ser executados continuamente.

No exemplo, a função `pinMode()` é usada para configurar o pino GPIO, onde se encontra o LED como saída. A função `digitalWrite()` altera o valor da saída entre `LOW` e `HIGH`. Observe que, para esse módulo, o LED acende quando a saída é configurada para `LOW`, em lógica negativa. A função `delay()` tem o objetivo de aguardar um tempo, contado em milissegundos.

8.1.2 Conexão WiFi

Conectar à rede WiFi é tão simples quanto na linguagem Lua. O código a seguir apresenta um exemplo:

```
#include <ESP8266WiFi.h>
const char* ssid = "ssid";
const char* senha = "senha";

void setup() {
    Serial.begin(115200);
    delay(10);

    // Iniciando a conexão WiFi

    Serial.print("Conectando para a rede");
    Serial.println(ssid);

    WiFi.begin(ssid, senha);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
```

```
    Serial.println("");
    Serial.println("WiFi conectado");
    Serial.println("Endereço IP: ");
    Serial.println(WiFi.localIP());
}
void loop() {
}
```

O código inclui funções específicas para o ESP8266, por isso, deve incluir o arquivo *ESP8266WiFi.h*. A conexão ao WiFi foi realizada por meio da função `setup()`, visto que pode ser feita apenas na inicialização do dispositivo. A função `WiFi.begin()` inicia a conexão WiFi, passando o SSID e a senha, se necessário. A função `WiFi.status()` retorna a situação da rede, sendo que `WL_CONNECTED` indica que a conexão foi feita com sucesso. Como ela não se conecta de forma instantânea, é importante aguardar em um laço até que a conexão seja efetuada. Para finalizar, a função `WiFi.localIP()` informa o endereço IP atribuído pela rede.

Esse código conta com mensagens de depuração que podem ser visualizadas apenas se a interface serial do microcontrolador estiver ligada. Normalmente, essa interface só é utilizada durante o desenvolvimento, para verificar o funcionamento dos programas. Para isso é necessário inicializar a interface serial usando o comando `Serial.begin()`, que recebe como parâmetro a velocidade da conexão. Após esse comando, é recomendado um atraso de 10 ms para seu processamento. As mensagens são enviadas pela interface serial usando o comando `Serial.print()` ou `Serial.println()`. Esse último adiciona uma quebra de linha no final da mensagem.

8.2 Servidor web

O exemplo a seguir mostra um servidor web que pode receber dois tipos de solicitações: para ligar, ou desligar, um LED e para fazê-lo piscar. Tem o mesmo efeito do exemplo correspondente no capítulo sobre programação com linguagem Lua. É possível observar as mudanças significativas na forma de programar o módulo com Lua ou no ambiente Arduino.

```
#include <ESP8266WiFi.h>
const char* ssid = "ssid";
```

```
const char* senha = "senha";
WiFiServer server(80); // Define a porta a ser usada
unsigned long MiliAnterior = 0;
int led = LOW;
int val = 0;
void setup() {
    Serial.begin(115200);
    delay(10);

    // Configura a porta D0 do LED
    pinMode(D0, OUTPUT);
    digitalWrite(D0, 0);

    // Iniciando a conexão WiFi
    Serial.print("Conectando para a rede");
    Serial.println(ssid);

    WiFi.begin(ssid, senha);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi conectado");
    Serial.println("Endereço IP: ");
    Serial.println(WiFi.localIP());
}

void loop() {
    if (val)// Pisca apenas quando a variável val está habilitada
        if(millis() - MiliAnterior >= 1000) { // 1 segundo
            MiliAnterior = millis();
            if (led == LOW)
                led = HIGH;  // Desliga o LED - Lógica invertida
            else
                led = LOW;   // Liga o LED
            digitalWrite(D0, led);
```

```
        }

    // Verifica se algum cliente conectou
    WiFiClient client = server.available();
    if (!client) {
        return;
    }

    // Espera até o cliente enviar sua requisição
    Serial.println("Novo Cliente");
    while(!client.available()) {
        delay(1);
    }

    // Lê a primeira linha da requisição
    String req = client.readStringUntil('\r');
    Serial.println(req);
    client.flush();

    // Verifica se um botão foi acionado

    if (req.indexOf("DESLIGA") != -1)
        val = 0;
    else if (req.indexOf("PISCA") != -1)
        val = 1;

    client.flush(); // Limpa o buffer do cliente

    // Envia resposta em formato HTML
    String s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n";
    s +="<!DOCTYPE HTML>\r\n<html><h1>Exemplo WEB - Pisca</h1>";
    s +="<p><a href=\"?com=PISCA\"><button><b>PISCA LED</b> </button>
</a>";
    s +="<p><a href=\"?com=DESLIGA\"><button><b>DESLIGA</b> </button>
</a>";
    s +="</html>\n";

    client.print(s);
    delay(1);
```

```
    Serial.println("Cliente desconectado");
  }
```

A maior diferença em relação aos programas apresentados até o momento está na forma de verificar o tempo de piscar o LED. A linguagem Lua trabalha com temporizadores associados a eventos. Assim, quando um temporizador atinge o tempo predeterminado, a função de tratamento é chamada. Já o ambiente Arduino não requer nenhum tipo de tratamento de evento. A temporização pela função `delay()` tem uma espera ocupada, ou seja, o processador não pode fazer nada enquanto está parado na função `delay()`. Se usarmos a função `delay()` para fazer o LED piscar, o processador não tratará outras requisições que ocorrerem durante a espera desse tempo, como, por exemplo, solicitações da rede.

Como não é possível, então, usar a função `delay()`, a base de tempo para fazer o LED piscar é calculada a partir do resultado da função `millis()`. Essa função retorna o tempo, em milissegundos, desde que o microcontrolador é ligado e "estoura", retornando a zero, após 50 dias.

8.2.1 Acesso ao banco de dados MySQL

Uma das grandes vantagens de utilizar o ambiente Arduino para o ESP8266 é a possibilidade de acessar bancos de dados MySQL com a biblioteca `mysql` do Arduino. Para aplicações em nuvem, esse acesso abre inúmeras possibilidades. Um dispositivo IoT pode alimentar diretamente um banco de dados na nuvem inserindo dados sensoreados e até consultando o estado que deve manter suas saídas.

O acesso a banco de dados não é uma tarefa simples. Envolve vários passos: i) boa modelagem; ii) mapeamento em linguagem SQL; iii) instalação e configuração do banco de dados; iv) manutenção e backup. Para uma aplicação de IoT, o ideal é usar algum banco de dados disponível na nuvem, evitando as tarefas iii e iv. No entanto não é seguro manter uma banco de dados aberto na Internet. O protocolo de acesso a banco de dados não provê a mesma segurança certificada do protocolo HTTPS, usado para a web. Assim, o mais comum é manter o acesso ao banco de dados encapsulado em uma interface web, como Web Services.

Exemplos mais adiante neste livro mostrarão o acesso a bancos de dados

por interfaces web. Essa seção usará um servidor de bancos de dados disponível na nuvem de forma gratuita: db4free.net. É um servidor para testes, no qual não há garantia da manutenção dos dados. O exemplo se aplicaria a qualquer servidor de banco de dados que pudesse ser instalado na intranet, por exemplo, o que limitaria os problemas de segurança de abrir seu acesso na internet.

O exemplo a seguir mostra como acessar um banco de dados MySQL. Antes de testar o código, é preciso instalar, configurar e criar um banco de dados em um servidor MySQL. O acesso pode ser feito por meio de um servidor instalado na rede local em que esteja o dispositivo IoT, ou em qualquer local da internet que esteja acessível. Alguns servidores MySQL estão disponíveis de forma gratuita na internet, como o db4free.net. O armazenamento é restrito a 200 MB. O *db4free.net* declara-se como banco de dados de testes, alertando que não deve ser considerado para operação.

A figura 8.3 apresenta informações sobre o banco de dados criado para esse exemplo.

Figura 8.3 – Interface de administração do banco de dados criado.

É necessário anotar **endereço, nome, usuário e senha** do banco de dados. Para o banco criado, o nome do banco foi `iotiot`, o usuário, `sergiool_iot`, e a senha, `iotiot256`. O servidor do banco de dados MySQL que será usado pode ser acessado pelo nome `db4free.net`.

A **administração do banco de dados** pode ser feita pelo próprio site: *db4free.net*. O site disponibiliza a interface de administração

PHPMyAdmin, versão web gratuita, para administrar bancos de dados MySQL. Com essa ferramenta, apresentada na figura 8.3, é possível visualizar dados e a estrutura do banco, criar tabelas e executar comandos SQL.

O exemplo do código para acesso ao banco de dados é mostrado a seguir. Um sensor de temperatura LM35 é ligado à entrada analógica `A0` e o valor lido da temperatura é registrado no banco de dados. Para isso, uma tabela foi criada no banco de dados contendo os seguintes campos: 1) `num`: inteiro autoincremental; 2) `valor`: ponto flutuante; 3) `hora`: valor temporal, registrado automaticamente na inserção. A tabela poderia ser criada pela interface web ou enviando o seguinte comando SQL para o banco:

```
CREATE TABLE sql3134961.registro_temp (
    num integer primary key auto_increment,
    valor float,
    recorded timestamp
);
```

O código Arduino:

```
#include <ESP8266WiFi.h>
#include <MySQL_Connection.h>
#include <MySQL_Cursor.h>
#define LM35 A0; // Define o pino que irá ler a saída do LM35

WiFiClient client;
MySQL_Connection conn(&client);

void setup() {
    IPAddress server_addr;
    Serial.begin(115200);
    delay(10);
    // Conecta à rede WiFi
    WiFi.begin("ssid", "senha");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.println(".");
    }
```

```
    // Imprime o endereço IP
    Serial.println(WiFi.localIP());

    // Obtém o IP do servidor, pois a conexão não aceita nomes...
    WiFi.hostByName("db4free.net", server_addr);

    Serial.println(server_addr);
    Serial.println("Conectando ...");
    // Ip do Servidor, porta, usuário e senha
    if (conn.connect(server_addr, 3306, "sergiool_iot", "iotiot256")) {
        delay(1000);
    } else
        Serial.println("Falha na conexão");
}

void loop() {

    MySQL_Cursor cur_mem(&conn);

    // SQL Exemplo
    double temperatura = (double(analogRead(LM35)))*0,322;
    char toStr[10];
    // SQL Exemplo
    String INSERT_SQL = "INSERT INTO iotiot.registro_temp (valor)
VALUES (";
    INSERT_SQL += dtostrf(temperatura, 3, 1, toStr);
    INSERT_SQL += ")";

    Serial.println("Gravando dados ...");

    // Executa a consulta
    cur_mem.execute(INSERT_SQL.c_str());
    // Libera memória. Importante devido às limitações dessa plataforma
    cur_mem.close();
    delay(2000);
}
```

O código inicia com a conexão WiFi, como os demais. No entanto observa-se que a variável que referencia a conexão com o MySQL,

`MySQL_Connection conn`, precisa ser instanciada passando a referência de um cliente de rede. Como o exemplo usa WiFi, uma variável do tipo `Wificlient` deve ser criada e passada como parâmetro. Essas variáveis são declaradas externas às funções, pois serão usadas nas funções `setup()` e `loop()`.

Após realizada a conexão WiFi, a conexão ao MySQL é preparada, obtendo-se o endereço IP do servidor, visto que a conexão não pode ser feita pelo nome, e, em seguida, a conexão é realizada com o nome de usuário e senha. A porta normalmente usada para acesso ao MySQL é a porta 3306.

Para enviar os comandos SQL ao banco de dados é necessário um objeto do tipo `MySQL_Cursor`, que recebe a referência à conexão e que tem o método `execute()` para enviar as consultas, escritas em SQL diretamente ao banco de dados. Nesse exemplo, apenas uma inserção foi feita.

A tabela do banco de dados usada tem três campos: uma chave primária, um inteiro autoincrementado; o registro da temperatura, com uma casa decimal; e o registro da hora e da data em que a informação foi enviada. Desses três campos, apenas o valor da temperatura precisa ser enviado. Os demais são registrados automaticamente pelo banco de dados.

O sensor de temperatura LM35 é linear, convertendo tensão para temperatura na proporção de 0,01 volt para cada 1o C. Assim, a 25º C, sua saída é igual a 0,25 volt. A entrada analógica do ESP8266 tem precisão de 10 bits, logo, assume os valores de 0 a 1.023 para tensões de entrada de 0 a 3,3 volts. Cada unidade na sua leitura equivale, então, à tensão de 3,3 V/1.024 = 0,0032 V. Logo, a cada 0,01/0,00322 = 3,10 unidades, tem-se 1º C. O valor em grau centígrado corresponde, então, ao valor lido dividido por 3,10 ou multiplicado por 1/3,10 = 0,322. Como a multiplicação é feita mais rapidamente, é melhor deixar a fórmula com a multiplicação.

O próximo trecho de código apresenta a função `loop()` sendo usada para uma consulta ao banco de dados usado no exemplo anterior. A função `setup()` permanece a mesma do exemplo anterior:

```
#include <ESP8266WiFi.h>
#include <MySQL_Connection.h>
#include <MySQL_Cursor.h>
```

```
WiFiClient client;
MySQL_Connection conn(&client);

void setup() {
    IPAddress server_addr;
    Serial.begin(115200);
    delay(10);
    // Conecta à rede WiFi
    WiFi.begin("ssid", "senha");

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.println(".");
    }

    // Imprime o endereço IP
    Serial.println(WiFi.localIP());

    // Obtém o IP do servidor, pois a conexão não aceita nomes...
    WiFi.hostByName("db4free.net", server_addr);

    Serial.println(server_addr);
    Serial.println("Conectando ...");
    // IP do servidor, porta, usuário e senha
    if (conn.connect(server_addr, 3306, "sergiool_iot", "iotiot256")) {
        delay(1000);
    } else
        Serial.println("Falha na conexão");
}


void loop() {

    // Inicia as variáveis
    MySQL_Cursor cur_mem(&conn);
    char SELECT_SQL[] = "SELECT * FROM iotiot.registro_temp";
    Serial.println("Consultando dados ...");

    // Executa a consulta
```

```
    cur_mem.execute(SELECT_SQL);

    // Obtém o cabeçalho das colunas
    column_names *cols = cur_mem.get_columns();
    for (int f = 0; f < cols->num_fields; f++) {
        Serial.print(cols->fields[f]->name);
        if (f < cols->num_fields-1)
            Serial.print('|');
    }
    Serial.println();

    // Obtém as linhas
    row_values *row = NULL;
    while (row = cur_mem.get_next_row()) {
        for (int f = 0; f < cols->num_fields; f++) { // Para cada linha
            Serial.print(row->values[f]);
            if (f < cols->num_fields-1)
                Serial.print('|');
        }
        Serial.println();
    }

    // Libera memória. Importante devido às limitações dessa plataforma
    cur_mem.close();
    delay(2000);
}
```

A resposta a uma consulta a um banco de dados retorna sempre um conjunto de tuplas que pode ser interpretado como uma tabela na qual cada linha corresponde a uma tupla. A biblioteca de acesso ao MySQL provê estruturas de dados para acesso direto aos dados dessa tabela.

Duas estruturas de dados semelhantes foram usadas para receber os dados provenientes do banco de dados. Os cabeçalhos da tabela podem ser recebidos pelo método `get_columns()`, e as linhas da tabela, pelo método `get_next_row()`, que deve ser chamado em um laço até que retorne nulo, indicando o final da tabela. Os resultados desses métodos são ponteiros para estruturas do tipo `column_names` e `row_values`, respectivamente. Ambos os métodos retornam um vetor cujo tamanho pode ser identificado pelo

atributo `num_fields` da estrutura `column_names`.

A execução desse código vai imprimir, pela interface serial, algo semelhante à listagem a seguir:

```
num|valor|hora
1|23.6|2016-09-13 20:37:59
2|23.6|2016-09-13 20:38:18
3|23.6|2016-09-13 20:38:37
4|23.6|2016-09-13 20:38:56
5|23.6|2016-09-13 22:18:06
6|23.6|2016-09-13 22:18:41
7|23.6|2016-09-13 22:18:59
8|23.6|2016-09-13 22:19:18
9|23.6|2016-09-13 22:19:37
10|23.6|2016-09-13 22:19:55
```

Dispositivos IoT não devem ter, a princípio, demandas para grandes consultas a bancos de dados, mesmo porque sua memória limitada não consegue manipular volumes maiores de dados. Mas pode ser útil para ler valores específicos a ser usados em suas configurações. Por exemplo, a temperatura-objetivo de uma aplicação de controle de temperatura pode estar registrada em um banco de dados, facilmente alterada por interfaces web ou smartphones, que pode ser lida pelos dispositivos IoT para configurar suas interfaces. É uma típica aplicação IoT na nuvem, na qual as interfaces de configuração são mantidas em servidores externos para facilitar seu acesso.

8.3 Atualização OTA

Uma característica bastante interessante para a manutenção dos sistemas embarcados é a possibilidade de atualização OTA, *Over the Air*, ou seja, sem fios e sem necessidade de desconectar o circuito. Isso possibilita que os sistemas no ESP8266 possam ser atualizados e corrigidos em produção. Ou seja, não é necessário recolher e atualizar todos os nós para uma correção ou atualização. É possível deixar os nós programados para um procedimento periódico de atualização no qual os microcontroladores podem verificar se houve uma atualização e, se houver, podem já se atualizar de forma automática.

Três tipos de atualização OTA estão disponíveis nas bibliotecas Arduino:

1. atualização pelo IDE Arduino usando a rede WiFi;
2. atualização pelo protocolo HTTP atuando como servidor;
3. atualização pelo protocolo HTTP atuando como cliente.

A IDE do Arduino só reconhecerá o módulo disponível para atualização se ele estiver na mesma rede local, visto que ele procura pelo protocolo mDNS. Nos métodos por HTTP, é possível fazer o acesso pelo IP possibilitando que este seja feito remotamente.

8.3.1 Atualização OTA pela IDE Arduino

Na atualização pela IDE Arduino, o ESP8266 permanece com uma porta aberta para atualização pela rede TCP/IP. A IDE Arduino pode se conectar à essa porta e enviar a atualização. Caso a IDE identifique a presença de um dispositivo com essa porta aberta, será apresentada a opção de usar esse dispositivo para atualização.

Uma biblioteca Arduino conta com os procedimentos necessários. E já conta também com um exemplo que deve ser incorporado para que a atualização pela rede seja possível. O exemplo segue abaixo, com algumas modificações.

```
#include <ESP8266WiFi.h>
#include <ESP8266mDNS.h>
#include <ArduinoOTA.h>

void setup() {
    int tentativas = 0;
    Serial.begin(115200);
    delay(10);
    // Conecta à rede WiFi
    WiFi.begin("ssid", "senha");

    while (WiFi.status() != WL_CONNECTED && tentativas++ < 10) {
        delay(500);
        Serial.println(".");
    }
    if (WiFi.waitForConnectResult() != WL_CONNECTED) {
```

```
        Serial.println("Falha na conexão! Reiniciando...");
        ESP.restart();
    }

    ArduinoOTA.onStart([]() {
        Serial.println("Inicio");
    });
    ArduinoOTA.onEnd([]() {
        Serial.println("\nFim");
    });
    ArduinoOTA.onProgress([](unsigned int progresso, unsigned int
total) {
        Serial.printf("Progresso: %u%%\r", (progresso / (total /
100)));
    });
    ArduinoOTA.onError([](ota_error_t erro) {
        Serial.printf("Erro[%u]: ", erro);
        if (erro == OTA_AUTH_ERROR) Serial.println("Falha de
autenticação");
        else if (erro == OTA_BEGIN_ERROR) Serial.println("Falha na
inicialização");
        else if (erro == OTA_CONNECT_ERROR) Serial.println("Falha de
conexão");
        else if (erro == OTA_RECEIVE_ERROR) Serial.println("Falha no
recebimento");
        else if (erro == OTA_END_ERROR) Serial.println("Falha na
finalização");
    });
    ArduinoOTA.begin();
    Serial.println("Pronto!");
    Serial.print("Endereço IP: ");
    Serial.println(WiFi.localIP());
}

void loop() {
    ArduinoOTA.handle();
}
```

O código é baseado no exemplo que vem com a IDE Arduino. Basicamente, ele atua na configuração dos parâmetros para a atualização.

O WiFi foi modificado para reiniciar caso não seja possível conectar, visto que a aplicação depende do WiFi funcionando. Após dez tentativas, se a conexão WiFi não for estabelecida, o sistema será reiniciado. Para isso foi usada uma função da API, `ESP.restart()`.

O serviço de atualização é informado para a rede por meio do protocolo mDNS (multicast DNS), que é apenas uma versão do DNS simplificada, em redes nas quais o DNS não está disponível, ou em redes nas quais o dinamismo destas demanda uma autoconfiguração, de modo que o próprio equipamento de rede pode informar sua presença e seus recursos disponíveis. Seu funcionamento está limitado à rede local.

Após a gravação desse código, a opção de gravação pela rede deve aparecer no menu **Ferramentas->Portas** da IDE Arduino. Caso esta opção seja selecionada, é possível atualizar o software do microcontrolador pela rede WiFi. Algumas restrições, no entanto, devem ser verificadas:

- o ESP8266 deve estar na **mesma rede** que o computador usado para a atualização;
- as atualizações devem **incluir o código do exemplo** para que as atualizações futuras também possam contar com esse recurso;
- o ESP8266 deve ter **memória disponível** para copiar o novo firmware sem substituir o antigo; durante a cópia, os dois devem estar presentes na memória;
- a versão do IDE Arduino deve ser a **1.6.7 ou superior**;
- o computador deve ter o **Python 2.7** instalado, ou versão superior do Python 2; versões do Python 3 não são compatíveis; a pasta do Python 2 deve estar no PATH do computador, opção que pode ser marcada na instalação do Python.

Após a atualização, o ESP8266 é reinicializado e o novo programa é copiado sobre o anterior, já iniciando por ele.

Com a opção de atualização pela rede habilitada no IDE Arduino, o Monitor Serial não vai funcionar, visto que ele tentaria o acesso pela rede, que não está disponível. Para acompanhar as mensagens de depuração, é preciso usar outro programa de acesso à porta serial, como o Putty ou o Hyperterminal.

8.3.2 Atualização servidor web

Outra possibilidade de atualização remota é usar o ESP8266 como um **servidor web** capaz de carregar o arquivo do programa e fazer a sua atualização. Essa opção é mais abrangente, especialmente se o programa for gerado em outra plataforma que não seja a IDE Arduino.

Um exemplo está disponível na IDE Arduino e pode ser usado para essa implementação. Ele usa a biblioteca `ESP8266HTTPUpdateServer`.

Segue o exemplo:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266HTTPUpdateServer.h>

ESP8266WebServer httpServer(80);
ESP8266HTTPUpdateServer httpUpdater;

void setup(void) {

    Serial.begin(115200);
    Serial.println();
    Serial.println("Iniciando...");
    WiFi.mode(WIFI_AP_STA);
    WiFi.begin("ssid", "senha");

    while(WiFi.waitForConnectResult() != WL_CONNECTED) {
        WiFi.begin("ssid", "senha");
        Serial.println("Falha no Wifi, tentando novamente...");
    }

    httpUpdater.setup(&httpServer);
    httpServer.begin();

    Serial.print("HTTPUpdateServer pronto! Vá para http://");
    Serial.print(WiFi.localIP());
    Serial.println("/update no seu navegador\n");
}
```

```
void loop(void) {
    httpServer.handleClient();
}
```

As exigências são menores para usar a atualização web. Qualquer versão da IDE Arduino pode ser usada e até mesmo outras ferramentas de desenvolvimento. O acesso pode ser feito pelo endereço IP. A figura 8.4 mostra a interface web recebida quando o acesso ao endereço indicado é realizado. Basta selecionar o arquivo e clicar em **Update**.

Figura 8.4 – Interface web utilizada para atualização OTA.

Deve ser observado, ainda, que o WiFi foi configurado para o modo `WIFI_AP_STA`. Nesse modo, o ESP8266 também atua como AP, e é possível conectar à rede WiFi mantida por ele, acessar o endereço *http://192.168.4.1/update* e atualizar o seu software.

8.3.3 Atualização como cliente web

O último método de atualização a ser apresentado para a plataforma Arduino é como **cliente web**. Nesse caso, cabe ao ESP8266 conectar-se a um servidor web remoto para buscar a atualização. É possível automatizar esse processo de forma que todos os módulos façam um acesso periódico a uma determinada URL para verificar a disponibilidade de novas atualizações. Caso uma nova atualização esteja disponível, ela pode ser automaticamente feita. Veja o exemplo a seguir:

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <ESP8266httpUpdate.h>
#include <EEPROM.h>

void setup() {
```

```
    Serial.begin(115200);
    delay(10);
    // Conecta à rede WiFi
    WiFi.begin("ssid", "senha");

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.println(".");
    }

    // Imprime o endereço IP
    Serial.println(WiFi.localIP());
    EEPROM.begin(4); // Inicializa a EEPROM para a versão
}

void loop() {
    HTTPClient http;
    http.begin("sergiool.esy.es", 80, "/versao.txt");
    // Início da conexão HTTP e envio do header
    if(http.GET() == 200) {
        uint8_t versao = EEPROM.read(0);
        uint8_t versaoSite = http.getString().toInt();
        Serial.printf("Versão Local: %d, Versão Remota: %d\n", versao,
versaoSite);
        if (versaoSite != versao) {
            EEPROM.write(0, versaoSite);
            EEPROM.commit();
            t_httpUpdate_return ret;
            ret =
ESPhttpUpdate.update("http://sergiool.esy.es/teste.bin");
            if (ret != HTTP_UPDATE_OK) {
                Serial.print("Falha na atualização: ");
                Serial.print(ESPhttpUpdate.getLastError());
                Serial.println(ESPhttpUpdate.getLastErrorString());
            }
        }
    }
    http.end();
}
```

Neste exemplo, um servidor web externo (*sergiool.esy.es*) disponibiliza dois arquivos: o arquivo do firmware, chamado *teste.bin*, e o arquivo *versao.txt*, que contém o número da versão atualmente em uso, representada por um número inteiro. A classe `HTTPClient` proporciona o acesso ao arquivo *versao.txt* para que se possa utilizar a versão atualmente carregada no site. A versão é registrada na memória EEPROM, uma memória não volátil, usada para armazenar dados permanentes, e esses dois valores são comparados. Para funcionar corretamente, o valor inteiro correspondente à versão deve ser atualizado no arquivo *versao.txt*, a cada atualização. Do contrário, o programa não será atualizado.

A função de atualização é a `ESPhttpUpdate.update()`, que reinicia o sistema logo após sua chamada. Portanto não é possível imprimir mensagens de depuração informando que a atualização foi realizada com sucesso. Caso isso ocorra, o sistema será reiniciado em seguida. Se houver algum erro na atualização, a mensagem de erro será enviada para a depuração.

8.4 Programando na IDE Eclipse

A IDE Arduino é muito limitada; não conta com vários recursos interessantes para os programadores, além de restringir alguns recursos da linguagem C++. Outras IDEs contêm recursos como completar automaticamente os métodos, dar opções de métodos a cada ponto digitado e, ainda, mostrar os parâmetros para cada método. Assim, é interessante conhecer alternativas à IDE Arduino para gerar código C++ para o ESP8266. Outros três ambientes estão disponíveis para compilar os programas para o ESP8266, usando todas as bibliotecas do Arduino: Platformio, Microsoft Visual Studio e Eclipse. Todas com versões gratuitas.

- **Platformio** (platform.io) é uma iniciativa recente de uma IDE integrada com várias linguagens e tecnologias, especialmente para sistemas embarcados. Ainda é pouco conhecida, mas tende a se tornar popular.
- **Microsoft Visual Studio** é o velho ambiente de desenvolvimento integrado da Microsoft, herdeiro, ainda, dos antigos Visual Basic e Visual C++. Sua versão “comunitária” é gratuita e integrável com o Arduino e com o ESP8266, mas é muito pesada. Para uma sensação de trabalho confortável, é necessário um computador com bom poder de

processamento e armazenamento rápido, o que não é o caso da maioria dos desenvolvedores.

- Finalmente, a **IDE Eclipse** tem ambiente de desenvolvimento agradável, é amplamente difundida, não causa grandes sobrecargas de processamento em um computador razoável e se integra bem com as bibliotecas Arduino e ESP8266. Creio que seja uma boa opção para os desenvolvedores de software para o ESP8266.

Existem duas opções para instalação do ambiente Arduino + Eclipse. A primeira utiliza um plugin do Marketplace da IDE Eclipse. Não requer instalação da IDE Arduino e é totalmente configurável pela IDE Eclipse. Para tanto, basta localizar o plugin no Marketplace, menu **Help** do Eclipse, e seguir alguns passos para configurar a IDE. Uma vez configurada, é possível compilar e gravar os módulos com o microcontrolador diretamente no Eclipse.

A segunda opção é mais fácil de configurar, visto que o responsável disponibilizou a IDE com todos os plugins já configurados. Assim, basta fazer o download em *http://eclipse.baeyens.it*, extrair os arquivos que estão em formato *.tar.gz*, usando o 7zip, ou outro aplicativo qualquer para descompactá-lo, e executar a IDE diretamente da pasta. Para habilitar o desenvolvimento sobre os módulos do ESP8266, é necessário adicionar o suporte a esse microcontrolador após a primeira execução. A opção está disponível pelo menu **Window → Preferences → Arduino → Platforms and Boards.** Em ambos os casos, é necessário ter o Java versão 8 ou superior instalado.

Apesar do trabalho adicional para configurar, a primeira opção tem um resultado mais limpo e orientado aos sistemas que se queiram desenvolver. Ambas as opções têm como resultado um ambiente de desenvolvimento mais profissional, com recursos que facilitam muito a vida do desenvolvedor. A figura 8.5 mostra como iniciar um projeto Arduino com o assistente do Eclipse. A seguir, a figura 8.6 mostra um projeto já em execução. A interface serial é apresentada à direita na tela, na parte inferior.

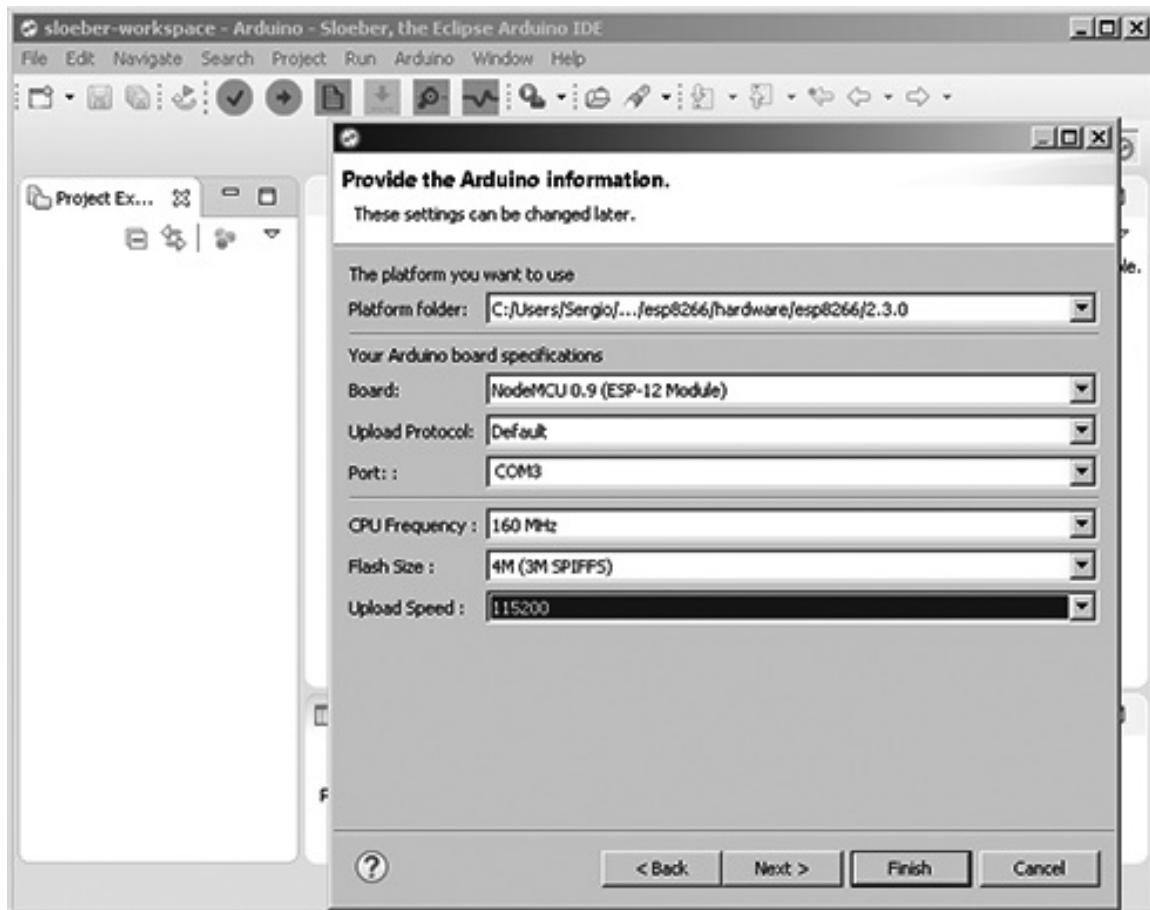



Figura 8.5 – Criando projeto Arduino na IDE Eclipse.

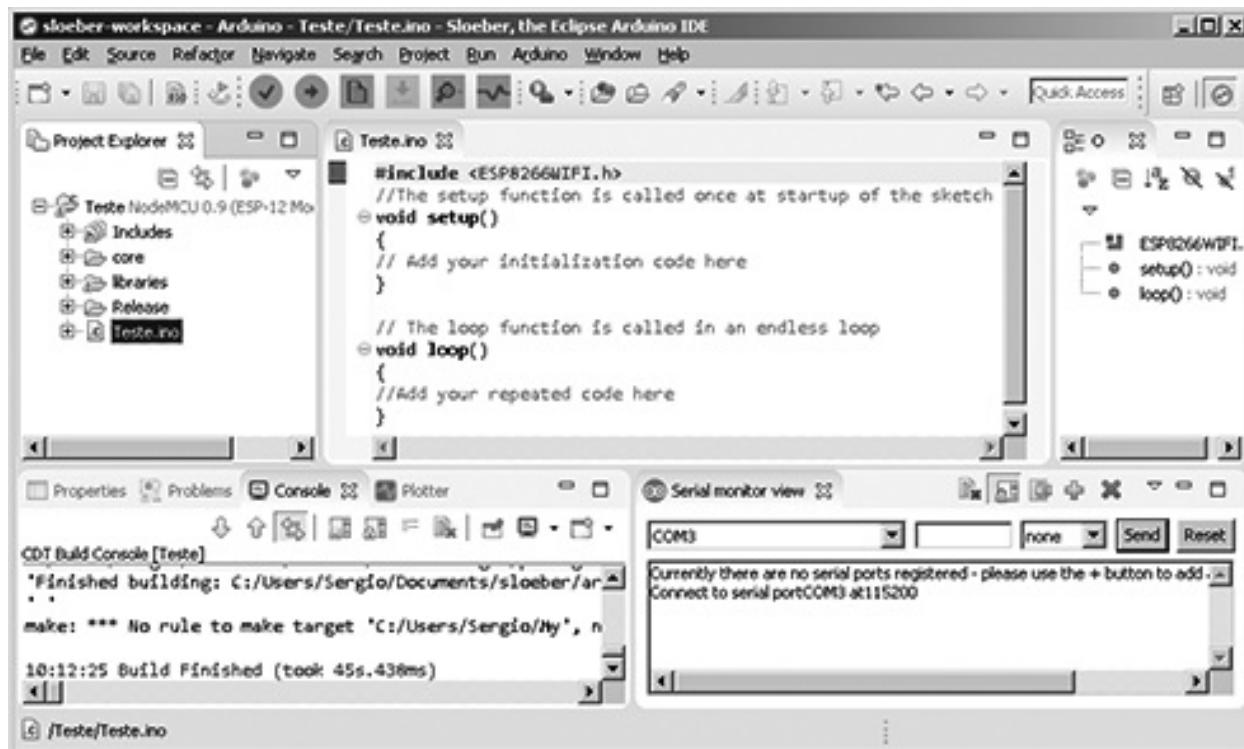



Figura 8.6 – Interface da IDE Eclipse desenvolvendo para Arduino.

A IDE Arduino é amadora, voltada apenas para curiosos em fase de aprendizado. A IDE Eclipse é focada em público profissional, para desenvolvimento de pacotes de software mais robustos e funcionais.

Há diversas vantagens em utilizar a IDE Eclipse.

- É possível incorporar qualquer biblioteca C/C++ padrão que seja compatível com a plataforma, mesmo que não tenha sido desenvolvida para Arduino. A IDE Eclipse é compatível com o padrão C++ em toda a sua extensão. A IDE Arduino tem restrições, como, por exemplo, não implementar os operadores de classes C++ `new()` e `delete()`.
- Eclipse é uma IDE com os recursos mais avançados para edição de código e depuração de erros. Engloba recursos como autocompletar para métodos, funções e parâmetros; identifica erros ao digitar, favorecendo o desenvolvimento de um código com menos erros de sintaxe e lógica; fecha parênteses, chaves e colchetes; gerencia melhor as bibliotecas, tanto do Arduino quanto as externas.
- A interface da IDE Eclipse é bem mais completa que da IDE Arduino. Inclui *Project Explorer*, que dá acesso aos arquivos do projeto, ao console, monitor serial e *Outline*, que possibilita ver as classes e funções presentes no código aberto.
- A IDE Eclipse suporta a inclusão de vários arquivos num projeto, o que possibilita criar projetos modulares com várias classes e arquivos.

Os próximos exemplos deste livro serão compilados e depurados usando a IDE Eclipse. A adoção dessa plataforma de desenvolvimento é fortemente sugerida para quem queira implementar para os dispositivos Arduino, em especial o ESP8266.

CAPÍTULO 9

Interface com sensores e atuadores

A IDE Arduino, bastante difundida entre entusiastas de microeletrônica, pode ser usada para programar uma vasta gama de placas e microcontroladores disponíveis a baixo custo. Em especial, os módulos que contêm o microcontrolador ESP8266 também podem ser programados com a IDE Arduino. Ela tem suporte à comunicação WiFi e inclui diversos periféricos, inclusive dezenas de módulos de sensores, atuadores, dispositivos com comunicação serial, muitos protocolos em modo servidor ou cliente, como web, MQTT e acesso a bancos de dados, entre outros.

Pouquíssimas aplicações podem ser previstas para microcontroladores sem outros componentes em suas interfaces de I/O ou de comunicação. A evolução da microeletrônica, aliada à capacidade produtiva da China com baixo custo, tornou disponíveis inúmeros componentes e módulos que podem ser integrados às diversas interfaces dos microcontroladores.

As interfaces mais comuns para a integração de módulos e componentes são: entradas e saídas digitais (GPIO), entradas analógicas, saída PWM e seriais (SPI, I2C e UART). Alguns módulos e componentes serão apresentados neste capítulo, agrupados de acordo com a interface utilizada.

9.1 GPIO

As interfaces de propósito geral de entrada e saída (GPIO) são extremamente comuns nos microcontroladores, disponíveis em grupos de 4 a 16 unidades digitais. O estado de cada pino de GPIO pode estar associado ao valor 0 ou 1. E as interfaces podem estar configuradas como entrada ou saída.

Caso uma interface esteja configurada como entrada, cada pino de GPIO

pode ser externamente associado ao valor 0 ou 1. Caso esteja configurada como saída, a tensão dos pinos GPIO será definida em software e alterada pelo microcontrolador. Os valores 0 e 1 são associados a tensões. Na saída, corresponde às tensões 0 e à tensão de trabalho do microcontrolador, 3,3 volts no caso do ESP8266. Quando o valor 0 é associado à tensão 0 volt e o valor 1 é associado à tensão nominal do microcontrolador, a lógica é dita **positiva**. Em alguns microcontroladores, a lógica pode ser invertida, sendo que o valor 0 é associado à tensão alta e o valor 1 à tensão 0 volt. Nesse caso, a lógica é dita **negativa**.

As interfaces GPIO podem ser usadas para ligar dispositivos de entrada simples como botões e chaves, ou dispositivos de saída do tipo liga-desliga, como LEDs ou relés, que podem ser usados para acionar motores, lâmpadas, resistências ou quaisquer tipos de dispositivos elétricos. Também é possível acionar dispositivos hidráulicos ou pneumáticos usando válvulas elétricas para o acionamento. Alguns dispositivos e sua forma de ligação serão mostrados a seguir.

9.1.1 LED

Os **LEDs** são usados para sinalização de situações específicas, como o estado ligado ou desligado para qualquer dispositivo ou periférico. Como são dispositivos de baixa corrente e sem resistência interna, requerem um resistor em série, em torno de 1 K, para limitar sua corrente. A figura 9.1 apresenta duas configurações possíveis para ligação de um LED a uma saída GPIO. Na primeira, o LED acende quando a saída apresenta tensão positiva. Na segunda, o LED acende quando a saída vai a 0 volt.

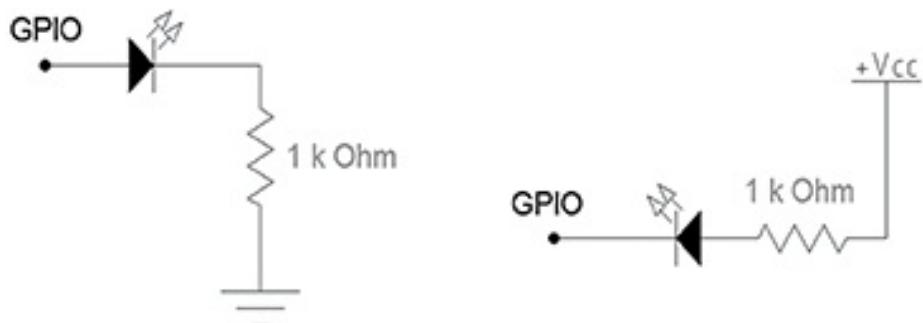



Figura 9.1 – Circuitos para a ligação de LED.

9.1.2 Relés

Para controlar dispositivos elétricos em geral, é preciso usar dispositivos de acionamento do tipo relé. A corrente de saída das GPIOs é muito baixa, e não é possível usá-las para acionar diretamente a maioria dos dispositivos, à exceção de LEDs ou dispositivos de baixíssima corrente. A figura 9.2 apresenta um módulo com dois relés, bem como o esquemático para um relé. Esse módulo conta com acoplamento óptico para evitar retorno do relé para a entrada GPIO, que poderia queimar a entrada. Também conta com um diodo invertido em paralelo com a bobina K, para zerar a contracorrente induzida pela bobina.

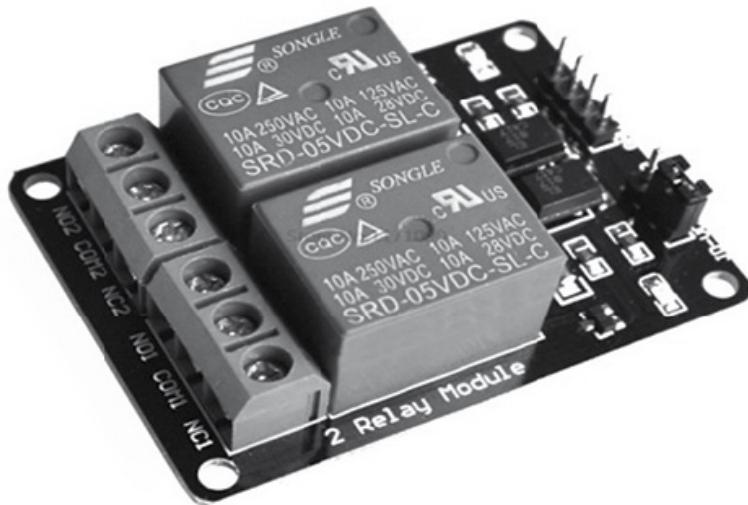



Figura 9.2 – Módulo com dois relés.

Qualquer tipo de equipamento elétrico pode ser ligado à saída do relé, desde que atenda às restrições, normalmente de 250 volts e 10 amperes para os modelos mais simples. Isso inclui motores, resistências, eletrodomésticos, lâmpadas e válvulas solenoides, para acionamentos hidráulicos. A figura 9.3 mostra o circuito interno do módulo com um relé. Ele tem acoplamento óptico e proteção de contracorrente. Em caso de curto, ou sobrecarga no equipamento, os danos não atingem o microcontrolador.

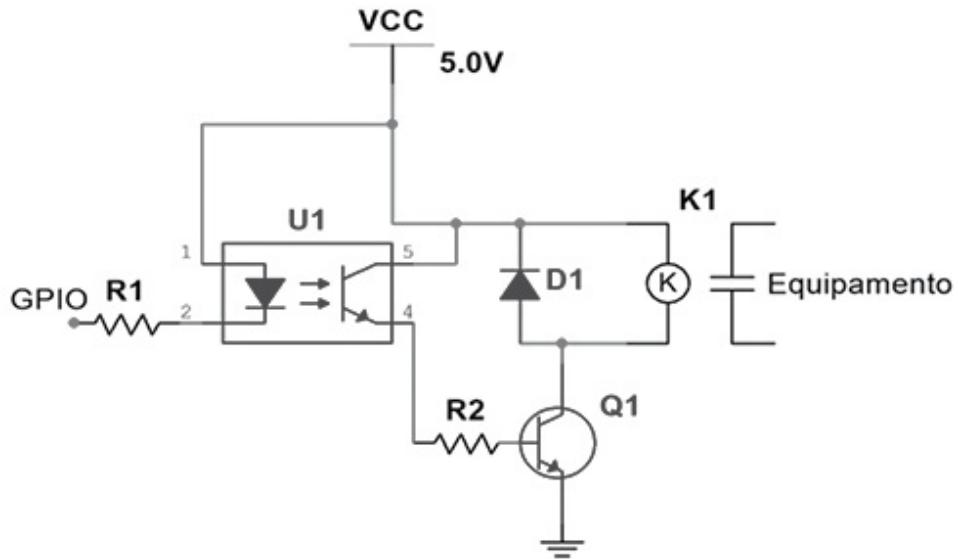



Figura 9.3 – Circuito do módulo com um relé.

Uma válvula solenoide simples é apresentada na figura 9.4. Esse modelo é relativamente barato, da ordem de poucas dezenas de reais. Pode ser usada em projetos de irrigação, reaproveitamento de água etc. Pode ser ligada diretamente à saída do relé, e o acionamento do relé acionaria o fluxo de água.

Figura 9.4 – Válvula solenoide de baixo custo.

9.1.3 Botões e chaves

Botões e chaves, ou interruptores, são úteis para receber um comando externo de um usuário, possibilitando o acionamento de dispositivos diversos. Os botões só mantêm o estado quando são acionados. As chaves

alternam seu estado entre os seus acionamentos. Sua ligação demanda um resistor para não deixar a entrada do microcontrolador em curto-circuito. Também é comum o uso de um capacitor em botões para prorrogar o estado do acionamento por tempo suficiente para que o programa possa efetuar a leitura da entrada correspondente.

Interruptores magnéticos também funcionam como chaves e podem ser acionados na presença de um ímã. São comuns para indicarem o fim de curso de portões ou se uma porta ou janela estiver aberta ou fechada. A figura 9.5 apresenta imagens de chaves magnéticas fim de curso (a) e porta aberta (b).

A figura 9.6 apresenta a ligação de um botão e uma chave a uma entrada GPIO. A presença do resistor é necessária para inverter a entrada. Sem o acionamento, o sinal 0 volt chega à entrada GPIO pelo resistor. Ao ser acionado, o sinal 3,3 volt chega direto ao pino GPIO, que consegue entender a mudança do sinal de entrada. A maioria dos sensores do tipo liga-desliga capacitivos, indutivos e magnéticos usa configuração parecida.

Figura 9.5 – Sensor fim de curso(a) e sensor de porta aberta (b).

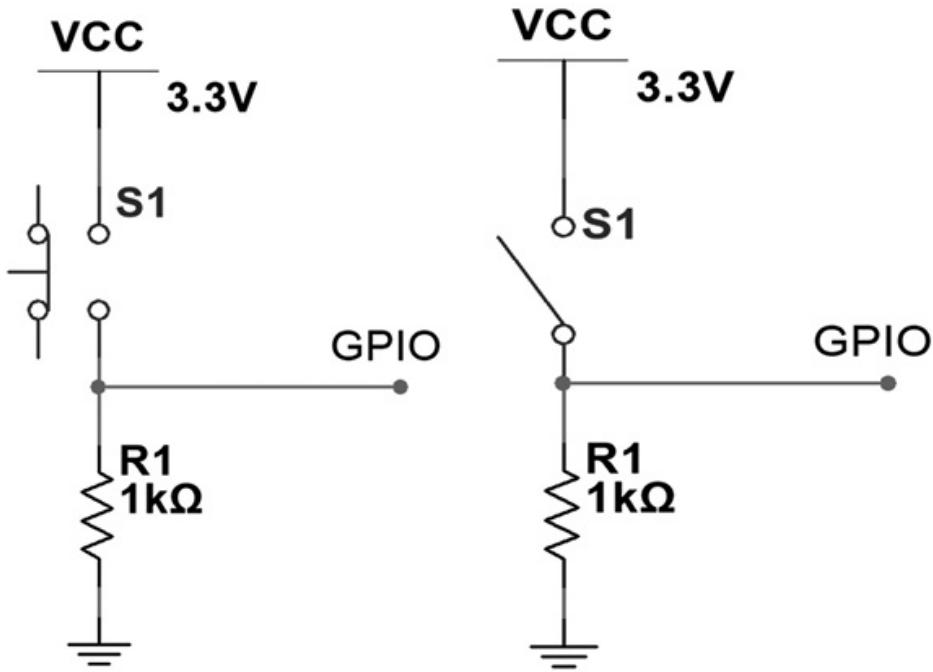



Figura 9.6 – Circuito para a ligação de um botão(a) ou uma chave(b).

9.1.4 Sensor de presença

O sensor de presença HC-SR501 é usado para identificar movimentos em distâncias até 7 metros. É normalmente usado em aplicações de alarme, acendimento automático de lâmpadas ou aberturas de portas automáticas. Seu funcionamento é digital e bastante simples: sua saída vai a 3,3 volts quando um movimento é detectado e permanece em 0 se não houver movimento. A saída permanece energizada durante um certo tempo, pois pode ser usada para acionar diretamente uma lâmpada ou manter uma porta aberta. A alimentação deve ser de 5 a 20 volts.

Há dois *trimpots* para ajudar a sensibilidade e o tempo em que a saída permanece com sinal alto. A sensibilidade mínima é de 3 metros, e a máxima, de 7 metros. O tempo mínimo é de cerca de 3 segundos, e o tempo máximo, cerca de 5 minutos.

O sensor de presença HC-SR501 pode ser visto na figura 9.7, com seus dois trimpots de regulagem.

Figura 9.7 – Módulo sensor de presença.

9.1.5 Controle remoto 315 MHz

Algumas aplicações exigem um acesso rápido e simples de forma remota, como o controle de abertura de um portão de garagem. Como o motorista tem as mãos ocupadas com a direção, ele pode preferir usar um controle remoto a ter de usar um smartphone para abrir seu portão. Além disso, o tratamento do sinal de um controle remoto direto tende a ser mais rápido do que um acesso via smartphone.

O controle remoto de 4 canais baseado nos ICs 2262 e 2272, como pode ser visto na figura 9.8, é uma opção barata e simples de se integrar a microcontroladores. Ele funciona na frequência aberta de 315 MHz e tem uma saída digital para cada canal que pode ser associada a um pino de GPIO.

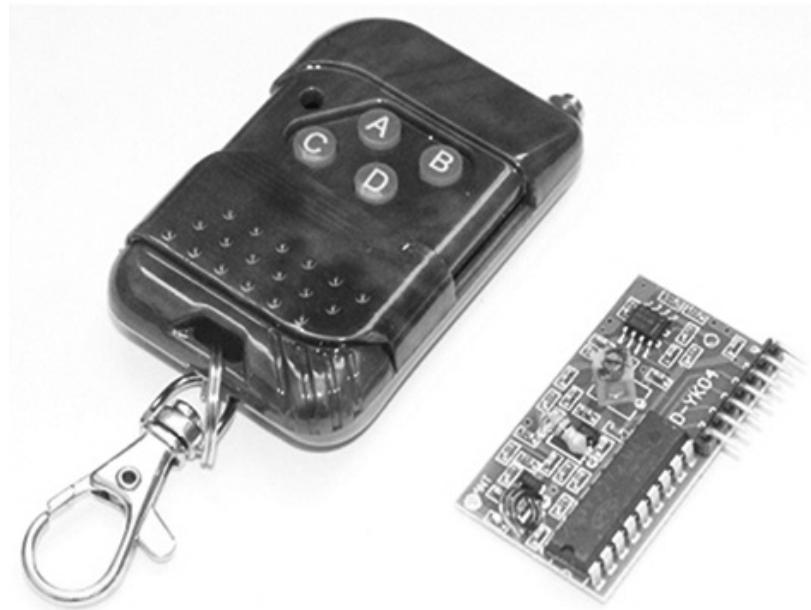



Figura 9.8 – Módulo de controle remoto com o controle.

9.2 Entrada analógica

A entrada analógica é usada para fazer leitura de dispositivos analógicos, cuja saída se dá em forma de tensão ou corrente variável. Essa entrada recebe uma tensão variável entre 0 volt e sua tensão nominal, convertendo essa leitura para um número inteiro, cuja precisão depende do número de bits reservado para a leitura.

O ESP8266 tem 10 bits de precisão para a leitura de sua porta analógica A0. Isso significa que podem ser representados valores entre 0 e 1.023, ou seja, 1.024 valores, para os quais 0 representa entrada igual a 0 volt e 1.023 representa sua tensão nominal: 3,3 volts. Cada unidade significa algo em torno de 3,2 mV, que seria, então, sua precisão mínima de leitura. Esse valor pode ser usado como referência para descobrir o valor da tensão lida. Multiplicando-se o valor lido por 3,2 mV chega-se à tensão lida.

Como a faixa de 0 a 3,3 volts não atende a todos os sensores analógicos, é comum tratar o sinal recebido antes de ligá-lo à entrada analógica. Em alguns casos, um divisor de tensão com dois resistores é suficiente. Em outros casos, pode ser necessário um amplificador operacional. São comuns sensores analógicos com saída especificada em corrente, entre 4 a 20 mA. Para ligar esses sensores, é necessário um conversor de corrente para voltagem. Existem módulos específicos para essa função, como o

módulo apresentado na figura 9.9. Ele converte a entrada de um sinal de 4 a 20 mA para a saída em 0 a 5 volts, que é a tensão mais comum entre os microcontroladores. Para o ESP8266, é possível regular a tensão de saída limitando a 3,3 volts em um trimpot de ajuste de ganho, ou fazer um divisor de tensão com dois resistores na saída.

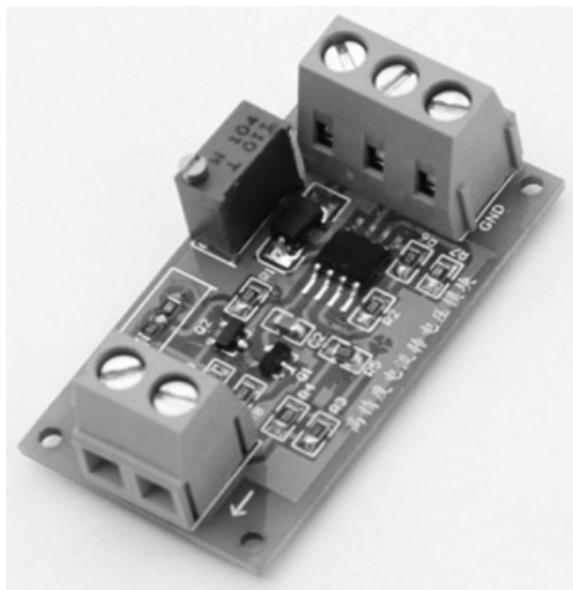
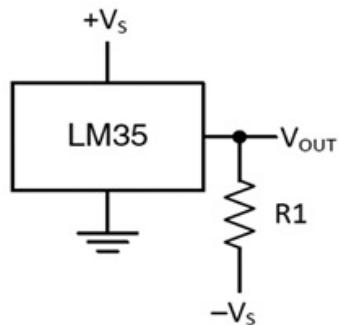



Figura 9.9 – Módulo conversor de 4-20 mA para 0-5 v.

A seguir, serão apresentados alguns sensores com interface analógica para exemplificar sua aplicação no ESP8266.

9.2.1 LM35

O sensor LM35 é um dos sensores de temperatura mais simples e comuns. Tem resposta linear, com 0,1 volt por grau Celsius. Sua faixa de operação vai de -55o C a 150o C. Sua tensão de saída varia linearmente de -0,55 a 1,5 volt. E sua alimentação deve ser entre 4 e 30 volts. Como o ESP8266, assim como os demais microcontroladores, não consegue ler tensões negativas, caso seja necessário ler temperaturas negativas, será preciso deslocar a tensão de trabalho do LM35.

Para uma leitura completa dos valores negativos, o terra do LM35 pode ser ligado à tensão de 0,55 volt, com um divisor simples de tensão. Assim, o terra do ESP8266 vai representar -0,55 volt para o LM35. Logo, a leitura 0 V vai representar essa temperatura mínima de -55oC. Como a tensão de trabalho do LM35 é de 4 a 30 volts, caso seja necessário alimentar seu terra

com 0,55 volt, sua tensão de alimentação deverá ser de, no mínimo, 4,55 volts. A figura 9.10 apresenta o circuito proposto no datasheet do LM35 para leitura de temperaturas negativas, com as referências de valores. A seguir, a figura 9.11 apresenta a imagem do LM35, que parece um transistor e custa algumas poucas unidades de reais.

Figura 9.10 – Referências para o LM35.

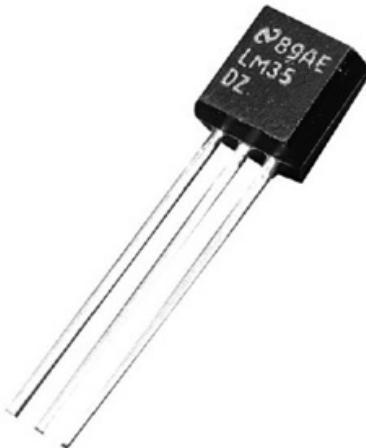



Figura 9.11 – Sensor de temperatura LM35.

Para utilização do LM35 no ESP8266, basta ligar seu pino de saída – Vout, na figura – ao pino A0 do ESP8266. O valor de tensão será lido como um inteiro entre 0 e 1.023, no qual cada unidade representa 3,2 mV. Como cada 0,1 V representa 1ºC, o valor lido deve ser multiplicado por 0,32 para se obter a temperatura lida.

9.2.2 Sensor de umidade do solo

Um sensor útil para aplicações agrícolas, o sensor de umidade do solo tem duas interfaces: uma digital e outra analógica. A interface digital pode ser usada para acionar um sistema de irrigação de forma automática, como uma válvula solenoide para a alimentação de água. A interface analógica pode ser usada para obter uma informação mais precisa a respeito da umidade do solo, apresentando essa informação como uma tensão de saída entre 0 volt e a tensão de entrada aplicada, na faixa entre 3,3 e 5 volts.

Como não existem unidades de medida de referência para umidade do solo, assim como ocorre com a temperatura, são necessários testes empíricos para obter mais precisão na leitura desse sensor. Um sistema de irrigação automático deverá considerar ajustes *in loco*, ao implantar o sistema, para que se obtenha a melhor configuração, de modo a manter a irrigação correta, sem desperdício de água e mantendo o solo suficientemente irrigado. Nesse sentido, a utilização de um sistema IoT com uma interface que possibilite ajustar os parâmetros da irrigação é de grande utilidade.

A figura 9.12 apresenta uma foto de um sensor de umidade do solo usado na irrigação. Há uma grande disponibilidade de fabricantes, mas, em geral, estes são baseados no CI LM193, um comparador de baixa voltagem que apresenta uma tensão de saída de acordo com a resistividade percebida nos eletrodos que deve ser tanto maior quanto maior for a umidade do solo. O tipo de solo, sua salinidade e seu pH também interferem na medida, o que é mais uma razão para que cada aplicação seja ajustada na implantação.

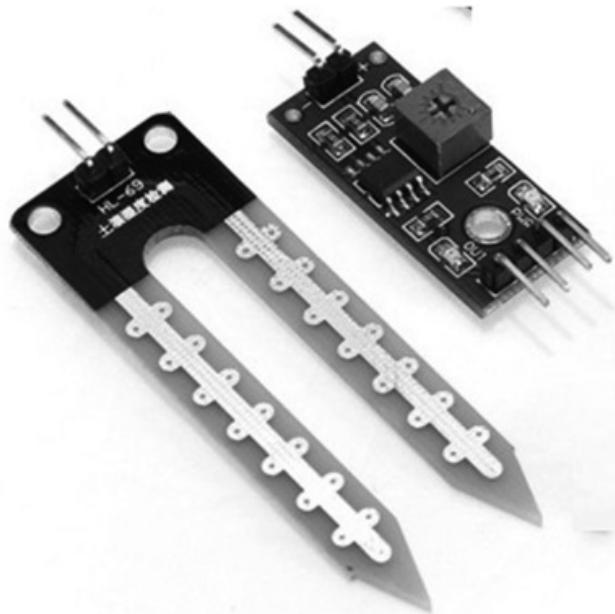



Figura 9.12 – Sensor de umidade do solo.

A leitura desse sensor pela porta analógica do ESP8266 vai resultar de um valor entre 0 e 1.023, no qual o valor 0 representa o solo seco e o valor 1.023 representa o solo encharcado. Para melhor entendimento do usuário, o ideal é dividir esse valor por 10,2; apresentando um pseudopercentual de umidade do solo variando entre 0 e 100%. E deve ser possível ao usuário ajustar os índices mínimos aceitáveis antes que o sistema de irrigação inicie seu trabalho, além do valor de referência para o qual ele deve encerrar a irrigação.

Deve ser observado, ainda, que é possível que a leitura do sensor suba rapidamente para o máximo, logo que o sistema seja ligado, pois um pouco de água na superfície do solo pode saturar o sensor. Por esse motivo é necessário manter o sistema de irrigação ligado por algum tempo, da ordem de minutos, para cada acionamento, mesmo que o sensor indique a saturação da umidade. À medida que a água penetra no solo, o sensor pode obter, novamente, leituras mais precisas sobre a umidade real do solo.

9.2.3 Sensor de gases inflamáveis

Um sensor que deveria estar presente em todas as residências, pois pode significar a diferença entre a vida e a morte, é o sensor de gases inflamáveis, em especial GLP e gás natural. O sensor responsável por essa detecção é o MQ-2, usado também em conjunto com um comparador, dessa vez, o

LM393, que tem também duas saídas semelhantes às do sensor de umidade do solo: uma digital e outra analógica. A saída digital indica a presença ou não de gases. A saída analógica apresenta um percentual representado por uma tensão entre 0 e 3,3 volts, da presença de gases, entre 300 a 10.000 ppm (partes por milhão), indicando a quantidade de partículas de gases presentes no ar.

A figura 9.13 apresenta uma foto do sensor. Além de GLP e gás natural, também detecta metano, propano e hidrogênio, gases também inflamáveis.

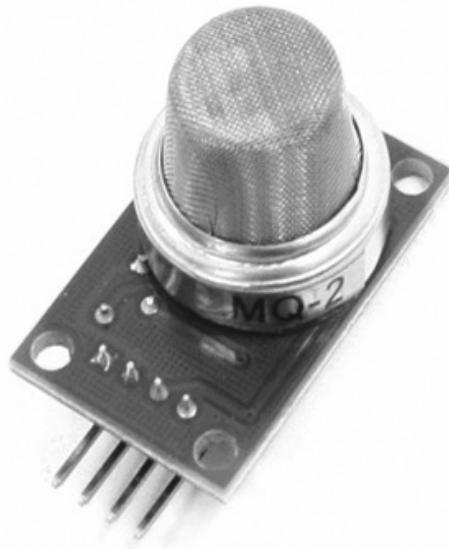



Figura 9.13 – Sensor de gases inflamáveis.

9.2.4 Sensor de corrente elétrica

Um sensor interessante para aplicações de automação residencial, e também industrial, é o sensor de corrente, como o módulo ACS712, que pode ser visto na figura 9.14. Esse módulo é capaz de informar a corrente que circular pelos seus terminais, alternada, entre 0 e 30 A (também há versões de 5 e 20 A, mas vamos considerar a versão de 30 A), ou contínua, entre -30 e +30 A, o que é muito útil para determinar se um determinado equipamento está ligado e qual o seu consumo. Para alguns equipamentos, como motores, o consumo pode, ainda, refletir sua carga e seu estado, visto que alguns tipos de motores podem aumentar a corrente caso tenham sobrecarga ou algum tipo de travamento.

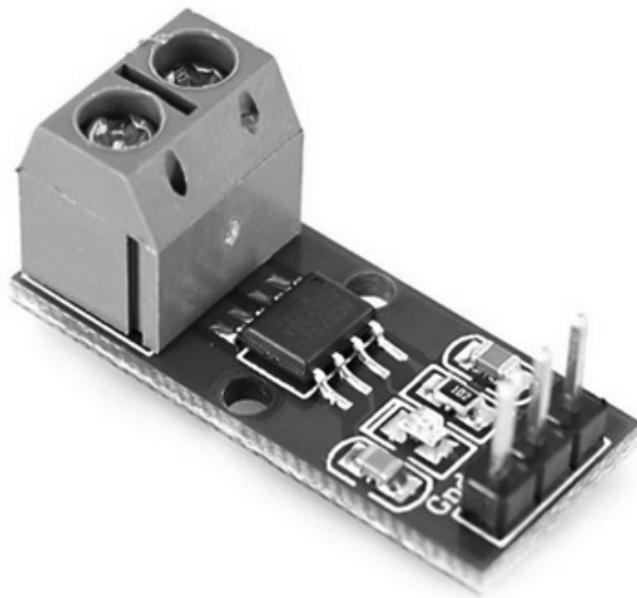



Figura 9.14 – Sensor de corrente.

O sensor ACS712 tem como saída uma interface analógica, de 0 a 5 volts, na qual cada 66 mV corresponde a 1 A, a partir de 2,5 volts. Assim, 0 A corresponde a 2,5 volts, 30 A é representado por 4,48 volts e -30 A corresponde a 0,52 volt. Como a entrada analógica do ESP8266 só consegue ler até 3,3 volts, é necessário fazer um divisor de tensão com dois resistores para reduzir a tensão máxima a 3,3 volts. Como a tensão máxima é 4,48 volts, é possível usar dois resistores para reduzir esse valor para 3,3 volts. Como sugestão, é possível citar os resistores 1,2 K e 3,3 K, cuja soma seria 4,5 K. Assim, o valor de 4,48 volts seria reduzido para 3,3 K com um erro de 0,4%, desprezível para esse tipo de sensor. Para essa redução de tensão, a correspondência a 1 A passa a ser de 48,4 mV. E o valor 0 A passa a ser representado por 1,83 volt. A figura 9.15 apresenta essa ligação.

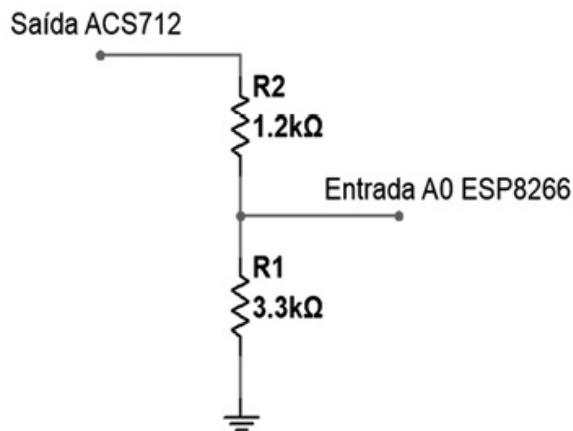



Figura 9.15 – Divisor de tensão para o ACS712.

A conta a ser feita, então, para se descobrir a corrente seria:

Tensão lida = valor lido na porta analógica / 1023 * 3.300 (em mV)

Corrente = (Tensão lida – 1.830) / 48,4

Para as demais versões desse sensor, deve-se observar que a versão de 5A utiliza o valor de 185 mV/A como referência e a versão de 20 A utiliza 100 mV/A. Os demais cálculos devem ser feitos considerando esses valores.

9.3 PWM

Ler uma entrada analógica é factível, visto que o microcontrolador só precisa ler uma tensão gerada por outros componentes, mas escrever uma saída analógica não é possível para microcontroladores. Como o microcontrolador é um dispositivo de potência muito baixa, ele não é capaz de gerar um sinal de tensão variada, e, mesmo que o fizesse, seria um sinal de potência muito baixa, o que não ajudaria no controle de dispositivos analógicos. Para acionar dispositivos analógicos, a alternativa viável é a utilização de um sinal de PWM.

O PWM (Pulse Width Modulation – modulação por largura de pulso) é uma técnica para controlar a potência de dispositivos ligados à saída de sistemas microcontrolados. Seu princípio básico é modular um sinal de onda quadrada sobre a alimentação do dispositivo. O PWM pode variar a largura de pulso para a onda quadrada quando ela se encontrar positiva, o que vai causar o efeito de ligar e desligar o dispositivo rapidamente, o que causa uma aparente queda de potência. A figura 9.16 mostra o funcionamento do PWM.

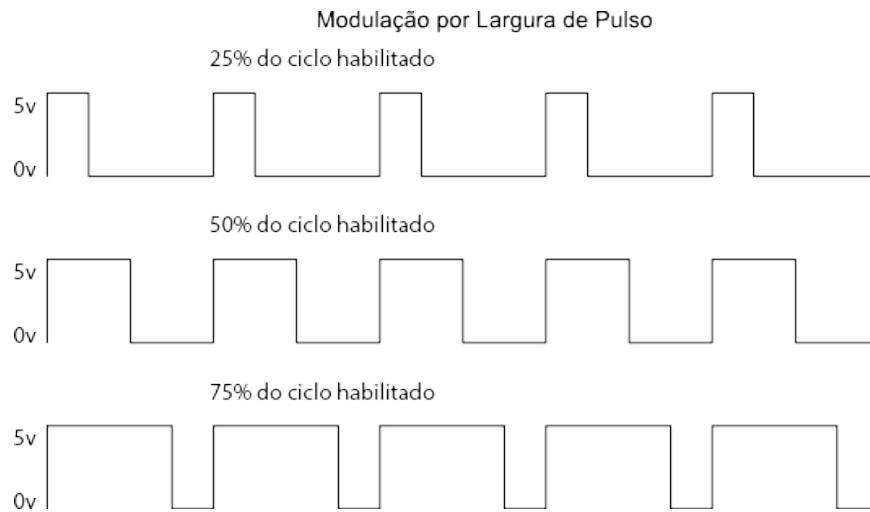



Figura 9.16 – Exemplo de saída PWM.

O PWM é mais eficiente que outros mecanismos de controle de potência como *dimer*, que associa uma resistência em série com a carga. Em série, a resistência consome parte da potência do circuito, indicada pela corrente que passa por ela. O PWM tem duas situações: ligado sem resistência ou desligado. Em nenhuma delas há consumo proporcional à corrente por parte do controle do PWM. Também é possível controlar a frequência do PWM. No ESP8266, a frequência-padrão é de 1 KHz, mas esse valor é configurável por software.

O PWM pode ser usado para controlar a temperatura da água de um chuveiro ou ebulidor sem consumir potência do sistema. Também é muito comum para controlar velocidade de motores. Um componente amplificador em sua saída pode ser usado para controlar equipamentos. O dimensionamento desse componente depende da potência do equipamento que se queira controlar. Pequenos motores podem ser acionados por simples transistores. Transistores de potências e TRIACs podem ser usados para cargas um pouco maiores. Cargas de alta potência podem ser controladas usando relés de estado sólido, componentes que podem ser chaveados em alta frequência, como é o caso do PWM.

A figura 9.17 apresenta um relé do estado sólido de 25 A. A figura 9.18, um TRIAC de potência de 40 A, componentes que podem ser usados para controlar cargas de alta potência como chuveiros e aquecedores. A vantagem do uso de TRIACs e relés de estado sólido sobre os relés comuns é o fato de não produzirem faíscas na abertura de contatos, garantindo,

assim, um tempo de vida maior, mesmo com a frequência do PWM. Relés comuns não podem ser usados com saídas PWM pois não funcionam na frequência de acionamento do PWM. Além disso, seu tempo de vida depende do número de acionamentos, o que se reduziria muito para frequências mais altas de acionamento.

Figura 9.17 – Relé de estado sólido.

Figura 9.18 – TRIAC.

No ESP8266, qualquer porta de GPIO pode ser usada como saída PWM. Assim como o Arduino, ele tem 10 bits de precisão, da mesma forma que a entrada analógica. Enviar o valor 0 para uma saída PWM representa zerar a largura de pulso positiva do PWM. Enviar o valor 1.023 representa 100% do tempo no pulso positivo. E, de forma linear, os percentuais variam de 0 a 1.023.

Como a entrada analógica tem a mesma precisão, é fácil construir um controle PWM baseado num potenciômetro. O potenciômetro pode ser ligado à entrada analógica do ESP8266, energizado entre 0 e 3,3 volts. Por software, o valor lido à entrada analógica pode ser enviado diretamente à saída PWM, e a largura do pulso, bem como a potência observada, será proporcional ao acionamento do potenciômetro.

O PWM também é usado para controlar a velocidade de motores usando ESC (Eletronic Speed Control – Controle Eletrônico de Velocidade), um dispositivo de controle de velocidade de motores bastante preciso, usado amplamente em aeromodelismo.

9.3.1 PONTE H L298N

O controle de motores com a ponte H L298N é um bom exemplo de uso do PWM. Com essa ponte, é possível controlar o sentido de rotação, bem como a velocidade de um motor de corrente contínua.

Uma ponte H é basicamente uma chave com quatro contatos e três posições. Considerando os contatos S1 a S4, a chave H tem uma posição com os contatos S1 e S3 fechados, S2 e S4 abertos; outra posição com S1 e S3 abertos, S2 e S4 fechados; e uma posição com todos os contatos abertos. Essa configuração possibilita inverter a rotação de um motor, visto que, num motor de corrente contínua, inverter sua polaridade inverte o seu sentido de rotação. Isso pode ser visto na figura 9.19.

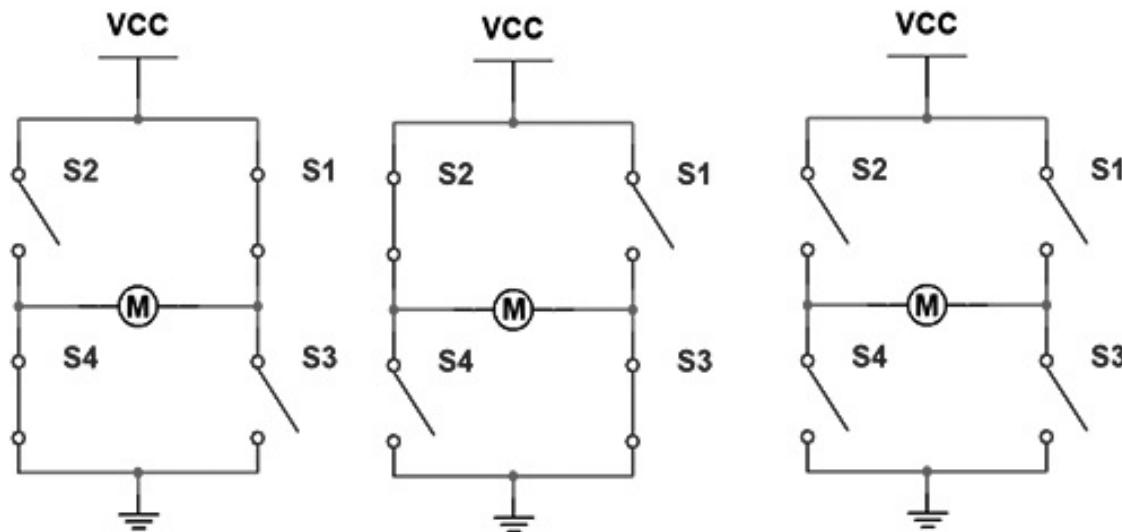



Figura 9.19 – Esquemático da ponte H L298N.

Além da rotação, a ponte H L298N também é capaz de alterar a velocidade de um motor, visto que ela é capaz de controlar a potência aplicada ao motor pelo PWM. Para tanto, basta ligar a entrada da ponte H a uma saída PWM do microcontrolador. A figura 9.20 apresenta uma imagem da ponte H L298N.

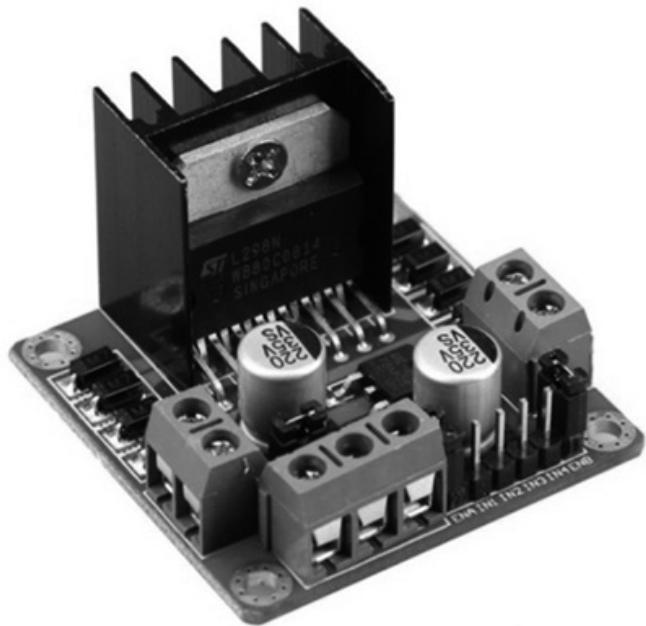



Figura 9.20 – Ponte H L298N.

9.4 Comunicação serial

Dispositivos mais complexos, capazes de apresentar mais informações e de forma mais precisa, normalmente usam interface de comunicação serial para se comunicar com os microcontroladores. A interface serial é digital e envia um fluxo de bits por um único pino. A frequência de operação pode ser síncrona, o que exige a utilização de um pino exclusivamente para o sincronismo, ou assíncrona, que não exige o sincronismo, mas tem velocidades mais limitadas, pois contém bits exclusivos para identificar o início e o fim dos sinais, além de uma codificação em dois ciclos para cada bit.

Os três tipos de comunicação serial mais comuns são SPI, I2C e UART. As duas primeiras são síncronas e a última, assíncrona. A interface UART é a mais simples de todas. Prevê apenas dois fios, TX e RX, e só possibilita a conexão ponto a ponto de um dispositivo. As interfaces SPI e I2C, além de

serem síncronas e proporcionarem maior velocidade, possibilitam a ligação de vários dispositivos na mesma interface. A interface SPI requer uma linha de seleção para cada dispositivo, habilitando a linha apenas do dispositivo para o qual vai se comunicar. Já a interface I2C trabalha com endereçamento, e os vários dispositivos podem utilizar as mesmas duas linhas, sendo assim, ela é mais compacta para a integração de vários dispositivos.

9.5 UART

A forma de comunicação mais simples e comum presente em quase todos os microcontroladores é a UART (Universal Asynchronous Receiver Transmitter), usada nas portas RS-232 (com tensão de 12 volts), muito comuns nos PCs antigos. É assíncrona e utiliza apenas dois pinos para comunicação, TX e RX, além da alimentação. No caso de sensores e outros dispositivos que não recebem nenhuma informação, apenas enviam, é possível usar somente um pino TX.

No início de cada transmissão são enviados sinais para sincronismo. A partir desses sinais, o receptor consegue identificar os demais bits a serem transmitidos.

A comunicação UART é normalmente usada para depuração dos programas dos microcontroladores, pois é popular nos PCs e, mesmo não existindo como item obrigatório dos PCs modernos, é normalmente implementada em uma interface USB. Assim, os programas escritos para os microcontroladores enviam as mensagens de depuração pela interface UART, sendo recebidas em programas próprios para comunicação serial como o Monitor Serial, do Arduino; Hyperterminal, do Windows; ou o popular Putty.

Os parâmetros de configuração da interface serial incluem a velocidade, normalmente até 115.200 bps; a presença ou não de bit de parada, controle de fluxo e bit de paridade. Para uma correta comunicação via UART, ambos os dispositivos devem estar configurados com parâmetros correspondentes.

9.5.1 Módulo GPS

Outro exemplo de uso da interface serial UART é para o GPS. Esses dispositivos, que se comunicam com a rede mundial de satélites para descobrir sua localização, utilizam uma interface UART para entregar essas informações a microcontroladores ou microcomputadores. Os dados são entregues em formato texto, em um protocolo conhecido como NMEA (National Marine Electronics Association), associação responsável pela definição desse protocolo. O quadro a seguir apresenta um exemplo do NMEA, que tem uma sintaxe bastante simples de ser interpretada.

```
$GPRMC,183729,A,3907.356,N,12102.482,W,000.0,360.0,080301,015.5,E*6F
$GPRMB,A,,,,,,,,,,,,V*71
$GPGGA,183730,3907.356,N,12102.482,W,1,05,1.6,646.4,M,-24.1,M,,*75
$GPGSA,A,3,02,,,07,,09,24,26,,,,,1.6,1.6,1.0*3D
$GPGSV,2,1,08,02,43,088,38,04,42,145,00,05,11,291,00,07,60,043,35*71
$GPGSV,2,2,08,08,02,145,00,09,46,303,47,24,16,178,32,26,18,231,43*77
$PGRME,22.0,M,52.9,M,51.0,M*14
$GPGLL,3907.360,N,12102.481,W,183730,A*33
$PGRMZ,2062,f,3*2D
$PGRMM,WGS 84*06
$GPBOD,,T,,M,,*47
$GPRTE,1,1,c,0*07
$GPRMC,183731,A,3907.482,N,12102.436,W,000.0,360.0,080301,015.5,E*67
$GPRMB,A,,,,,,,,,,,,V*71
```

Diversas bibliotecas estão disponíveis em código aberto, como o TinyGPS, desenvolvido para o Arduino, que pode ser facilmente integrável a qualquer projeto.

A figura 9.21 apresenta um módulo GPS compatível com o Arduino, bem como o ESP8266. Os pinos a ser usados são: VCC, GND, RX e TX.

Figura 9.21 – Módulo GPS.

9.6 SPI

A interface serial SPI (Serial Peripheral Interface) é um protocolo simples, síncrono, que define dois papéis: um lado atua como mestre e o outro como escravo. O mestre gera e envia o sinal de sincronismo. A cada pulso do sinal de sincronismo, um bit é enviado do mestre para o escravo e outro do escravo para o mestre. São usados quatro pinos:

- **CLK (ou SCK)** – *Clock* ou sincronismo;
- **MOSI** – *Master Out Slave In*, saída do mestre para o escravo;
- **MISO** – *Master In Slave Out*, saída do escravo para o mestre;
- **SS (ou CS)** – *Slave Select*, define qual escravo pode se comunicar com o

mestre.

É possível conectar vários dispositivos escravos em um barramento comum com um único mestre. Nesse caso, o mestre deve definir com qual escravo ele vai se comunicar.

A interface SPI é muito usada para ligar dispositivos de memórias, como leitores de cartão SD, ou memórias adicionais. Também é usada para ligar sensores que coletam mais de uma informação, como, por exemplo, o SCP1000, sensor barométrico que indica a pressão atmosférica. Outro exemplo comum de interface SPI é o módulo leitor de RFID MFRC522 da empresa NXP.

No ESP8266, os pinos para ligação da interface SPI são: SCK → D14, MOSI → D13, MISO → D12 e SS → D15. É possível configurar a interface SPI até 80 MHz, o que daria uma taxa de transferência da ordem de 80 mbps.

9.6.1 Módulo cartão de memória SD

Para ilustrar a interface SPI, um bom exemplo é a integração de um cartão de memória SD. Com esse cartão, é possível aumentar a capacidade de armazenamento do microcontrolador para a ordem de gigabytes, tornando possível sua utilização como servidor de arquivos, com protocolos FTP ou HTTP. Essa não é a principal vocação de um microcontrolador, tampouco pode-se esperar um bom desempenho nessa função, mas pode ser interessante em aplicações remotas, nas quais arquivos temporários precisam ser armazenados, ou mesmo drivers e aplicativos necessários para o acesso a outras funções.

O módulo cartão SD pode ser visto na figura 9.22. Podem ser vistos os pinos de interface CS, MOSI, SCK e MISO, além dos pinos para terra e alimentação, que pode ser de 3,3 ou 5 volts. Esse módulo, integrado com o ESP8266, pode atingir velocidades de transmissão da ordem de 10 MB/s. Não é alto, comparado à velocidade de leitura de memórias em PCs, mas é bem significativo, considerando que se trata de um microcontrolador.

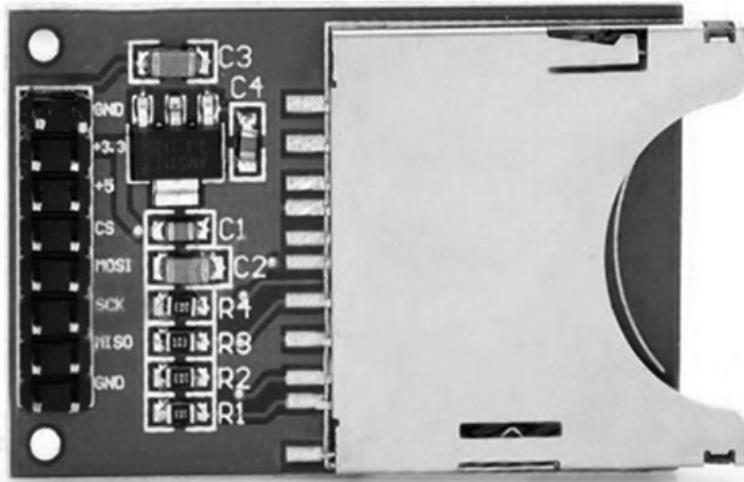



Figura 9.22 – Módulo de cartão de memória SD.

9.6.2 Módulo RFID RC522

O conceito de Internet das Coisas começou com RFID, e é esperado que se integrem as tags inteligentes a diversos projetos de RFID. Para tanto, o leitor mais popular de RFID é o RC522, que tem interface SPI e bibliotecas e vários exemplos de integração disponíveis para o ambiente arduino que podem ser compilados e carregados no ESP8266.

A figura 9.23 apresenta o módulo RFID-RC522 com duas tags que normalmente acompanham a placa como exemplo, um cartão e um chaveiro.

Na ligação ao ESP8266, é preciso considerar que os pinos associados às GPIOs 12 a 14 contêm uma implementação em hardware para a interface SPI. Logo, esses pinos devem ser ligados da seguinte forma: SCK → GPIO14, MOSI → GPIO13, MISO → GPIO12. Já os pinos SDA (CS ou SS) e RST podem ser ligados a quaisquer pinos GPIO do ESP8266, exceto o GPIO 15. Esse pino é ligado ao terra na inicialização normal, e os pinos CS e RST devem ser ligados ao nível alto para a inicialização e o funcionamento do módulo. Se o CS ou o RST estiverem ligados ao GPIO 15, o ESP8266 não será inicializado.

Figura 9.23 – Módulo leitor RFID.

9.7 I2C

A comunicação serial I2C (Inter-Integrated Circuit) foi desenvolvida pela Philips e é usada para integrar periféricos de baixa velocidade a microcontroladores e barramentos de microprocessadores. A Intel deu sua contribuição ao projeto com o SMBus (System Management Bus), um subsistema da I2C que define protocolos para o funcionamento de dispositivos nessa interface.

A interface I2C prevê apenas duas vias bidirecionais: SDA (Serial Data) para dados seriais e SCL (Serial Clock) para o sincronismo, também serial. O microcontrolador assume a função de mestre do barramento e os demais dispositivos ligados a ele assumem a função de escravo. O diferencial dessa interface é seu sistema de endereçamento, possibilitando que cada periférico escravo ligado a ele assuma um endereço e que todos se comuniquem no mesmo barramento.

Diversos dispositivos utilizam a interface I2C para se comunicar com os microcontroladores, desde módulos com vários sensores, como o GY-80 (Figura 9.24), sensor que agrega acelerômetro, magnetômetro, temperatura e pressão, a interfaces de cristal líquido, LCD e OLED, bem como relógios de tempo real (RTC).

Figura 9.24 – Módulo acelerômetro, magnetrômetro e sensor de temperatura e pressão.

9.7.1 GY-80

O módulo de sensores GY-80 agrega acelerômetro de três eixos, giroscópio de três eixos, magnetômetro e sensor de temperatura e pressão. Ele funciona com uma interface serial I2C. Para acessar o sinal desses sensores é necessário fazer uma varredura pelo barramento I2C para descobrir quais endereços estão disponíveis e correspondem a um sensor. Existem programas específicos para a varredura do barramento I2C para descobrir os endereços disponíveis, como o I2C Scanner, desenvolvido para o ambiente Arduino.

9.7.2 RTC DS1307 e DS3231

Um relógio de tempo real (RTC – Real Time Clock) é importante para todas as aplicações que requerem referência de tempo, visto que os microcontroladores não têm nenhum tipo de relógio interno capaz de registrar data e hora de forma contínua. Um RTC é capaz de manter e fornecer ao microcontrolador a data e a hora com precisão razoável. O modelo DS1307 tem precisão inferior, cerca de 14 ppm (partes por milhão), e o DS 3231 tem precisão de 2 ppm. O erro de 1 ppm significa um

minuto de atraso a cada milhão de minutos, o que dá aproximadamente dois anos.

A figura 9.25 apresenta o RTC DS1307.

Figura 9.25 – Relógio de tempo real DS1307.

Os relógios DS1307 e DS3231 se comunicam com os microcontroladores usando o protocolo I2C. Para esses dispositivos, há bibliotecas disponíveis para o ambiente Arduino, de fácil integração.

Aplicações que têm conectividade com a Internet podem usar servidores NTP (Network Time Protocol), um protocolo usado para disponibilizar informações temporais, para ajuste e acerto de relógio, com servidores disponíveis de forma gratuita em vários sites. A utilização do protocolo NTP dispensa o uso de relógios de tempo real.

9.7.3 Sensor de temperatura e umidade DHT11 e DHT22

Os sensores DHT11 e DHT22 identificam temperatura e umidade do ar. Eles têm, internamente, um pequeno microcontrolador de 8 bits que transmite as informações sensoriadas por um protocolo específico que utiliza apenas um pino para dados. Essa característica os torna muito interessantes, pois, além de este microcontrolador ser preciso, ele só ocupa

um pino digital, que pode ser qualquer pino de GPIO.

O formato dos dados inclui uma sinalização específica para o início da transmissão dos dados, e, em seguida, 40 bits, que representam 16 bits para a temperatura, 16 bits para a umidade e 8 bits para o controle de erros (*checksum)*. O Arduino conta com uma biblioteca específica para a leitura desses sensores que contêm as funções `readTemperature()` e `readHumitidy()`.

Os sensores DHT11 e DHT22 têm poucas diferenças. O primeiro é mais barato. O segundo é mais preciso e atende em uma baixa maior. Enquanto o DHT11 lê temperaturas entre 0 e 50ºC, com 2ºC de precisão, e umidade entre 20% e 80% com 5% de precisão, o DHT22 identifica temperaturas entre -40ºC e 125ºC, com precisão de 0,5ºC e umidade de 0 a 100%, com precisão entre 2% e 5%. O protocolo é o mesmo para ambos, bem como as bibliotecas disponíveis.

A figura 9.26 mostra o sensor DHT11.

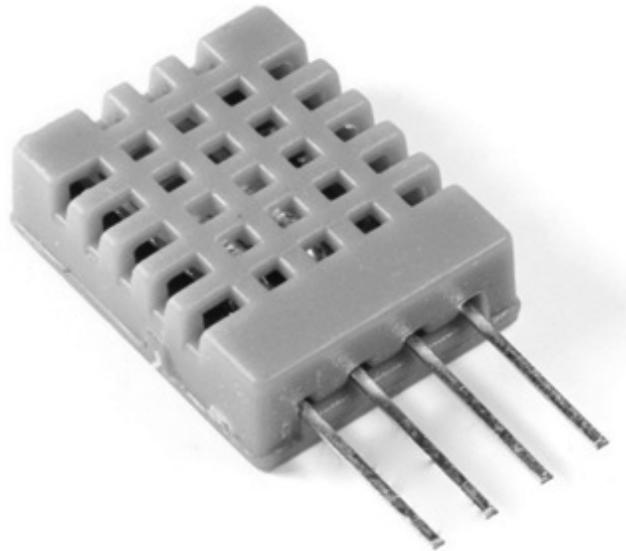



Figura 9.26 – Sensor de temperatura e umidade DHT11.

A disponibilidade de módulos sensores, atuadores e de comunicação disponíveis para aplicações IoT não se resume a estes aqui apresentados. A lista é extensa e inclui módulos com diversas especificidades. Para cada aplicação, um módulo pode ser selecionado. É preciso ficar atento à sua forma de comunicação com o processador – GPIO, analógica ou serial –, à disponibilidade de bibliotecas e aos requisitos gerais de tensão, temperatura de trabalho, faixa de operação etc. Escolher um módulo

periférico que atenda à aplicação é um dos fundamentos principais de Internet das Coisas.

PARTE III

Aplicações

CAPÍTULO 10

Datalog na nuvem

Este capítulo apresenta uma aplicação simples de registro de dados de temperatura no site ThinkSpeak.com. Um módulo com o microcontrolador ESP8266, ligado a um sensor de temperatura, envia periodicamente os dados de temperatura para o site usando a API específica para a IDE Arduino. E os dados, incluindo gráficos de registro de temperatura, podem ser visualizados diretamente pela web.

Uma aplicação típica de IoT é funcionar como *datalog*, cujo objetivo é apenas registrar uma grandeza obtida por um sensor. Esse registro pode ser usado com vários objetivos, desde monitoramento ambiental a aplicações industriais. Também é possível disparar alarmes ou ações baseadas nos valores registrados.

Datalog já foi um processo manual, passou a ser automatizado com armazenamento local, depois por um servidor em rede. O momento atual é de enviar o registro para a nuvem. Enviar para a nuvem economiza toda a infraestrutura necessária para o registro local e, de quebra, facilita o acesso remoto. Fazê-lo com o ESP8266 significa uma economia ainda maior.

Diversos serviços de registro de informações IoT estão disponíveis; alguns, de forma gratuita. Eles oferecem interfaces elaboradas, incluindo gráficos e aplicativos para acesso em smartphone. O mais popular desses serviços, no momento, é o *ThingSpeak.com*.

10.1 ThingSpeak.com

ThingSpeak.com é um serviço de IoT na nuvem que possibilita fazer registro de informações recebidas pelos dispositivos IoT, além de disponibilizar diversas análises e ações sobre esses dados. As análises incluem análise de dados por softwares matemáticos, visualização de

gráficos indicativos do comportamento dos dados e plugins para criar arquivos HTML, JavaScript ou CSS personalizados. As ações incluem o envio de *tweeters*, que podem ser encaminhados para comandar outras plataformas, além de controles temporizados, reações a eventos e comandos via protocolo HTTP.

Alguns exemplos de aplicações são citados no site: um sistema para contar carros usando uma câmera e um software para reconhecimento de imagens em um Raspberry Pi, uma estação meteorológica, um sistema de alerta de enchentes e um sistema de sincronismo de cores de luzes ligados em rede social. Em todos os exemplos, o site *ThingSpeak.com* atua como intermediário remoto, armazenando e disparando ações de acordo com os dados recebidos.

As interfaces disponíveis no *ThingSpeak.com* são intuitivas e agregam valor às aplicações. A figura 10.1 apresenta o monitoramento de um sistema de aquecimento solar de água. As grandezas monitoradas são a temperatura do painel solar e a temperatura do boiler. Componentes apresentam a temperatura atual e gráficos apresentam o histórico de temperatura.

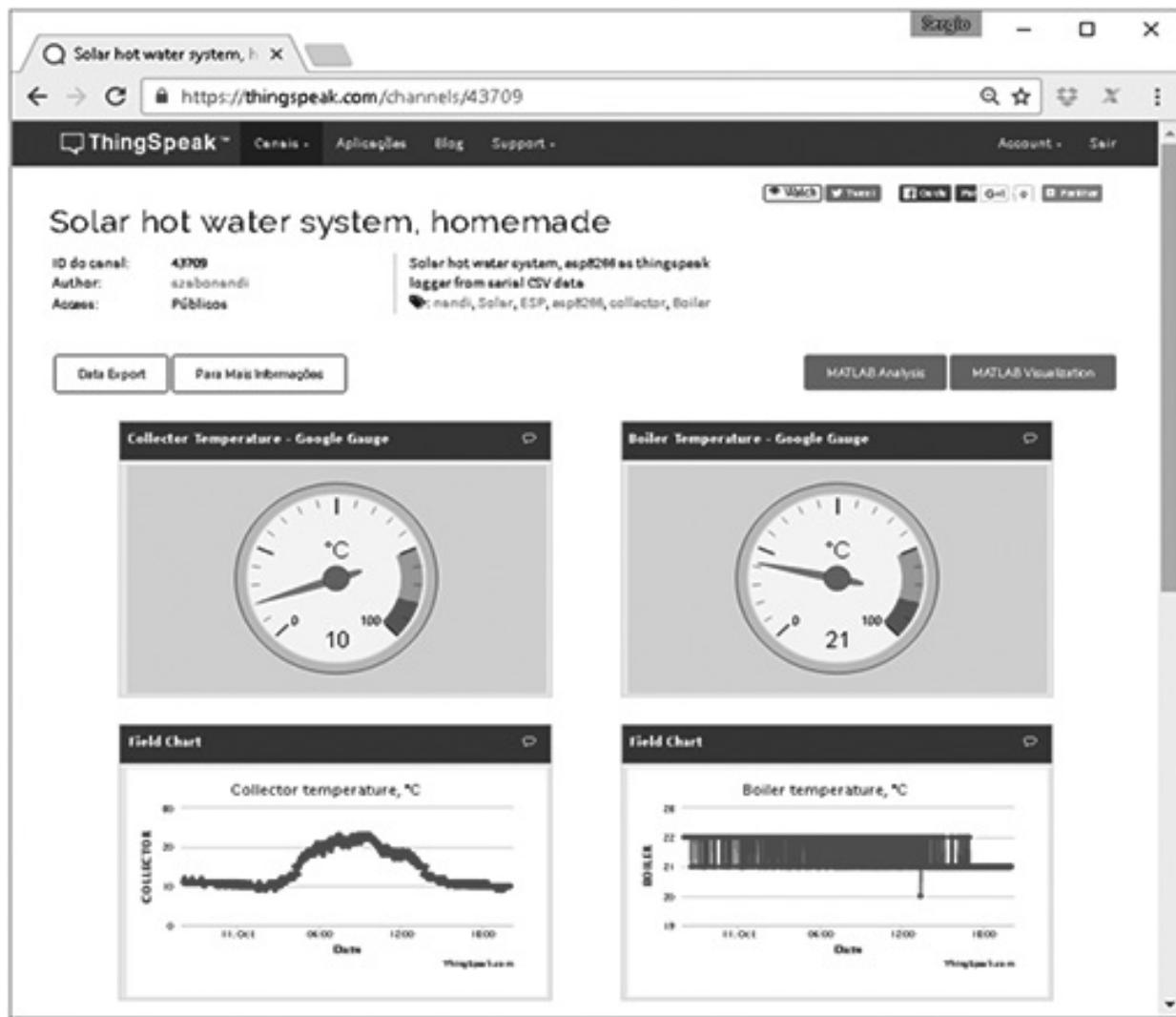



Figura 10.1 – *Interface exemplo do Thingspeak.*

Os controles e gráficos também podem ser acessados pelo aplicativo *Thingview*, disponível para smartphones Android.

10.2 Desenvolvimento de aplicações

O desenvolvimento de aplicações de IoT usando o *ThingSpeak.com* se inicia com o cadastro pelo site. Em seguida, deve-se cadastrar um canal. Ao canal será atribuído um identificador (*ID*) e uma chave para utilização restrita do canal. A esse canal é possível atribuir vários campos, bem como associar coordenadas e uma URL de referência da aplicação.

Para acessar o canal do Arduino e do ESP8266, uma biblioteca está disponível (`thingspeak-arduino`) e deve ser instalada. O arquivo *ThingSpeak.h* deve ser incluído para acesso à biblioteca.

Como exemplo, será apresentado um datalogger da tensão e da corrente de uma bateria ligada a um painel solar e a uma carga. O objetivo é monitorar a carga da bateria, bem como registrar em quanto tempo ela se carrega por dia e quanto se aproveita da carga. Essas informações são importantes para o dimensionamento de painéis e baterias.

O circuito com o ESP8266 vai monitorar a tensão da bateria e a corrente de carga/descarga da bateria. A tensão do painel solar será a mesma da bateria quando estiver em carga e será insignificante quando não houver carga, durante período sem sol. A tensão é monitorada por um divisor de tensão, visto que ela pode chegar a 15 volts no pico do painel solar. A corrente é monitorada usando o sensor de corrente ACS712. Ambos têm saída analógica, o que gera um problema adicional, visto que o ESP8266 só tem uma entrada analógica.

Para efetuar a leitura de duas entradas analógicas é possível fazer a multiplexação das duas em uma única entrada analógica do ESP8266. A multiplexação pode ser controlada por uma ou duas portas digitais. Para esse exemplo, duas portas GPIO são usadas para multiplexar os sinais do divisor de tensão e do sensor de corrente. Ao serem ligadas, elas habilitam a condução dos respectivos transistores, ligando-os à terra e anulando a entrada correspondente. Se ambas as portas GPIO de controle estiverem ligadas, a entrada analógica do ESP8266 deverá receber tensão 0 V.

A figura 10.2 apresenta o diagrama elétrico do circuito. À esquerda, têm-se o sensor de corrente ACS712, ligado à alimentação de 5 volts. Sua saída pode chegar a 4,48 volts. Como a entrada do ESP8266 limita-se a 3,3 volts, um circuito divisor de tensão é ligado à saída Vout do sensor, com objetivo de reduzir a tensão, proporcionalmente, à 3,3 volts, cobrindo toda a faixa de corrente sensoriada.

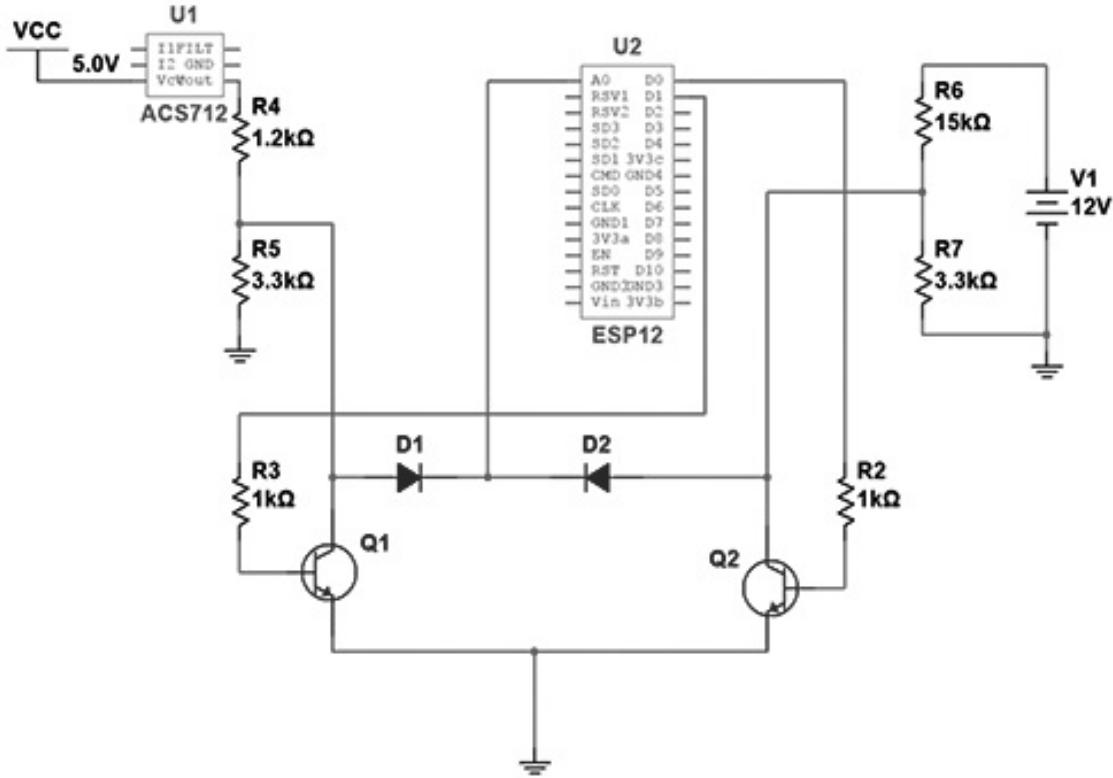



Figura 10.2 – Circuito para monitoramento de tensão e corrente.

À direita do circuito, encontram-se os resistores que funcionam como divisores de tensão para medir a tensão da bateria. A tensão nominal é de 12 volts. A tensão de carga não deveria passar de 15 volts, mas o painel solar é capaz de gerar até 20 volts, quando em aberto. Para garantir uma faixa de sobretensão acima dos 15 volts, os resistores divisores de tensão reduzem até 18,3 volts para 3,3 volts. Esses valores fracionados se devem à disponibilidade de resistores comerciais em valores que atendiam às faixas de tensão desejadas.

Dois transistores Q1 e Q2 se apresentam em configuração de chaveamento. Quando energizados, eles ligam a saída do sensor de corrente ou do divisor de tensão da bateria, respectivamente, à tensão 0 V. Essa configuração possibilita isolar o sensor de corrente ou o divisor de tensão da bateria para que o valor do outro possa ser lido pela entrada analógica. Os diodos D1 e D2 evitam que a tensão de 0 V atinja o outro lado, inutilizando também a outra entrada.

Assim, para ler a corrente, o programa deve enviar a configuração de saída alta (3,3 V) para o pino D0, o que vai ligar o transistor Q2, zerando a tensão que vem do divisor de tensão da bateria. A saída do pino D1 deve

ser mantida baixa (0 V) para que o transistor Q1 permaneça sem conduzir, o que mantém a tensão do sensor de corrente ligada à entrada analógica A0. E, assim, o valor de A0 por ser lido. Para ler a tensão da bateria, de forma análoga, a saída D0 deve ser mantida baixa (0V) e a saída D1 deve ser mantida alta (3,3V).

A figura 10.3 apresenta a montagem do circuito no protoboard. Os transistores são de uso geral, bem como os diodos. Para o teste, foram usados transistores BC548 e diodos 1N4002.

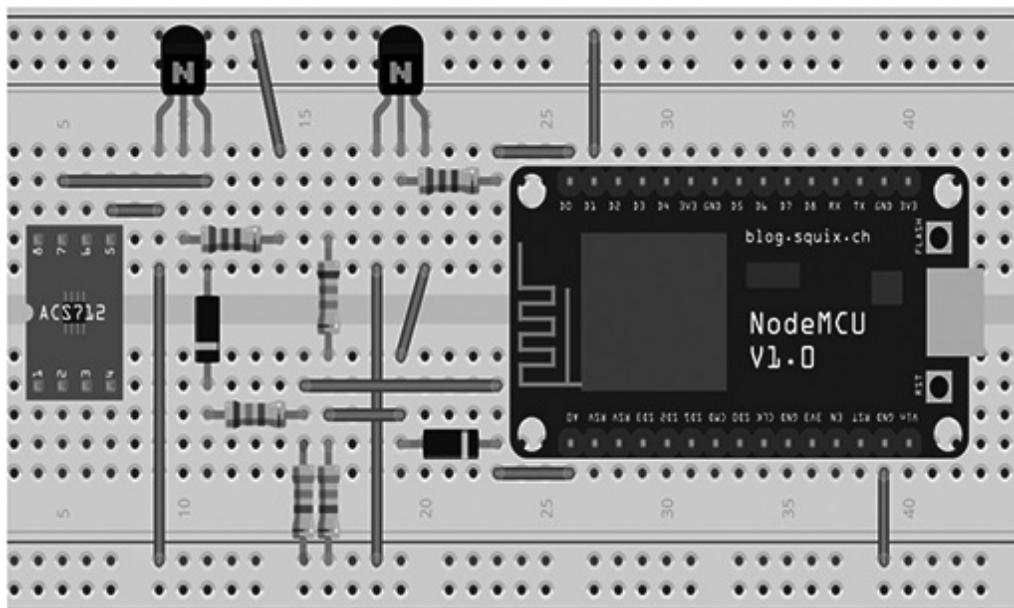



Figura 10.3 – Montagem do circuito.

10.3 Programa

Para obter a tensão lida efetivamente da bateria é preciso associar o valor 1.023 à tensão de 18,3 volts, que será convertida nos 3,3 volts, máximo possível da entrada analógica do ESP8266. Como o comportamento da tensão é linear, de 0 a 18,3 volts, basta uma regra de três simples para obter a tensão lida = valor lido *x* 18,3 / 1.023.

Para obter a corrente lida pelo sensor ACS712, é preciso usar as fórmulas de conversão para esse sensor, apresentadas na seção 9.4.2. Sendo:

Tensão lida = valor lido na porta analógica / 1023 * 3.300 (em mV)

Corrente = (Tensão lida – 1.830) / 48,4

Lembrando que a corrente pode ser positiva quando a teria estiver

alimentando a carga, ou negativa, quando a bateria estiver sendo carregada. A corrente máxima lida será de 30 A, o que significa que esse circuito é capaz de medir cargas de até 360 W, mesma potência para o painel solar a ser utilizado.

```
#include <ESP8266WiFi.h>
#include <ThingSpeak.h>
WiFiClient cliente;
const unsigned long NumeroCanal = 169736;
const char * ChaveEscrita = "42F62ANNENZUXZOM";
void setup() {
    Serial.begin(115200);
    delay(250);
    Serial.println();
    Serial.println("Iniciando....");
    WiFi.begin("ssid", "senha");
    int tentativas = 0;
    while ((WiFi.status() != WL_CONNECTED) && tentativas++ < 20) {
        delay(500);
        Serial.print(".");
    }
    ThingSpeak.begin(cliente);
}

void loop() {
    float sensor, tensao, corrente;
    // Ler a corrente D0 = 1 e D1 = 0
    digitalWrite(D0, 1);
    digitalWrite(D1, 0);
    sensor = analogRead(A0);
    tensao = sensor / 1023.0 * 3300.0;
    corrente = (tensao - 1830.0) / 48.4;
    ThingSpeak.writeField(NumeroCanal, 1, corrente, ChaveEscrita);
    // Ler a tensão da bateria D0 = 0 e D1 = 1
    digitalWrite(D0, 0);
    digitalWrite(D1, 1);
    sensor = analogRead(A0);
    tensao = sensor * 18.3 / 1023.0;
    ThingSpeak.writeField(NumeroCanal, 2, tensao, ChaveEscrita);
```

```
    delay(20000); // ThingSpeak aceita atualizações a partir de 15s
}
```

CAPÍTULO 11

Controle de irrigação com MQTT

O protocolo MQTT foi projetado para aplicações de Internet das Coisas. É leve, utiliza um servidor denominado *broker*, que pode estar hospedado na nuvem ou no local, e que repassa as informações entre os diversos dispositivos envolvidos, que podem contar com aplicações web, smartphone ou dispositivos IoT. Neste capítulo, é apresentado um exemplo de sua utilização para um sistema de controle de irrigação distribuído com interface em linguagem JavaScript para executar em um navegador web.

11.1 Protocolo MQTT

O protocolo MQTT foi idealizado pensando no ambiente de IoT para exemplos como esse apresentado na seção anterior. Tem as vantagens de ser padronizado e interoperar com um número grande de plataformas, entre gratuitas e pagas.

O protocolo MQTT, apresentado na figura 11.1, tem três tipos de elementos, a saber:

- **MQTT Broker** – Servidor que deve permanecer sempre disponível; faz a intermediação entre a geração de dados e sua publicação. Existem diversos brokers públicos disponíveis na internet. Caso se queira, é possível fazer o download de um broker MQTT gratuito para executar internamente na própria rede. Chama-se *Mosquitto* e está disponível em *mosquitto.org*.
- **Publisher MQTT** – Dispositivo IoT que gera a informação e envia os dados para o broker MQTT.
- **Subscriber MQTT** – Assinante do serviço disponibilizado pelo *publisher* MQTT. Deve receber as informações geradas pelo *publisher* pelo acesso

ao broker. Pode executar ações em função dessas informações ou disponibilizar uma forma de visualizá-las, como uma interface web ou smartphone.

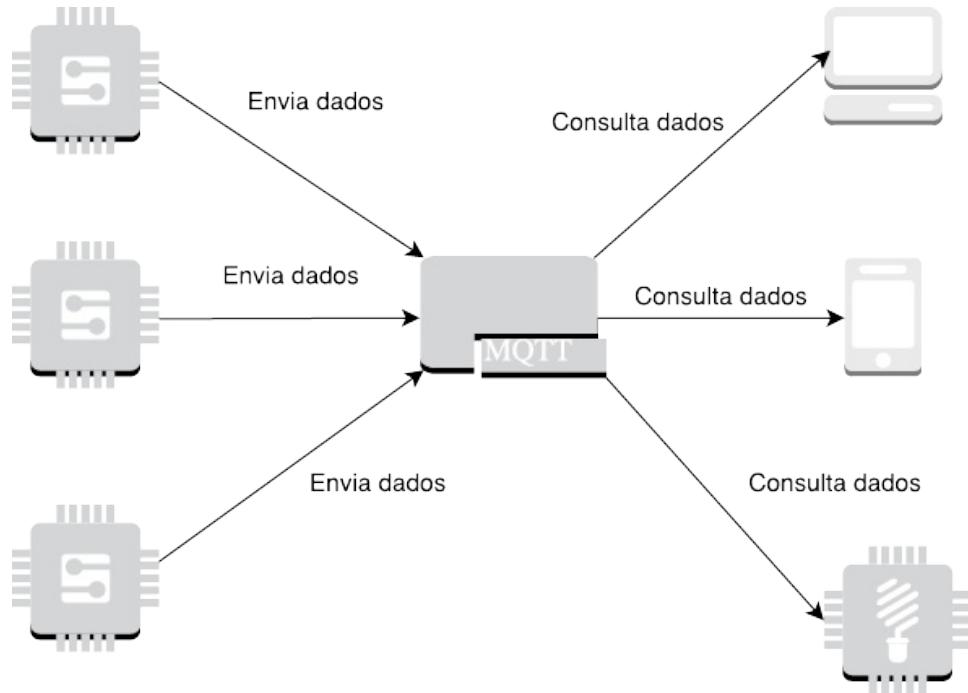



Figura 11.1 – Funcionamento básico do MQTT.

O protocolo MQTT foi desenvolvido para ser um protocolo M2M (*Machine-to-machine*), de dispositivo para dispositivo. Dessa forma, não foi concebido com nenhum tipo de interface-padrão ou forma de visualização ou análise. Cabe à aplicação de apresentação decidir como expor essas informações. Há interfaces para apresentação em aplicativos para smartphone, web ou desktop.

As aplicações mais comuns de utilização do protocolo MQTT incluem também registros do tipo *datalog*, além de acionamentos simples como acionar uma porta GPIO ou aumentar a velocidade de um motor usando PWM. Vale lembrar, nesse caso, que o protocolo não tem especificações de garantia de qualidade para aplicações de tempo real. Logo, é possível que os acionamentos demorem até alguns segundos para ser efetivados depois que forem solicitados por interfaces diversas.

Outra utilidade do protocolo MQTT é promover a comunicação entre dois dispositivos IoT. Apesar de existir inúmeras formas de fazê-lo diretamente, a utilização do protocolo MQTT é tão simples que evita a necessidade de

conhecimento prévio de endereços ou abertura de portas em roteadores e firewalls, visto que ambos os dispositivos vão atuar como clientes.

11.2 Implementação com MQTT

Para compreender o funcionamento básico do MQTT, será apresentado um exemplo que consiste na comunicação entre dois módulos ESP8266 e um broker para registro dos dados e comunicação. Os módulos atuarão em conjunto para determinar quando irrigar, além de fazer o acionamento da bomba de irrigação. Os dados, disponíveis no broker, podem ser acessados por aplicativos para smartphones bem como desenvolver aplicações em várias linguagens: Java, C/C++, Python, JavaScript, entre outras.

Para essa aplicação de irrigação serão considerados dois tipos de módulos básicos: um módulo com o ESP8266 para controlar uma bomba d'água, normalmente ligada próxima à caixa-d'água ou ao poço, que tem a função de verificar o nível com um sensor-boia; e um módulo próximo à área a ser irrigada, que tem a função de verificar a umidade do solo, com um sensor de umidade de solo, e acionar uma válvula solenoide para ligar o fluxo de água.

Os módulos de controle de bomba e umidade do solo podem ser replicados, se necessário, conforme a aplicação. Em alguns casos, por exemplo, além de uma única caixa-d'água, podem existir várias, ou pode existir um poço, cujo nível deve também ser verificado para evitar que a bomba seja ligada sem carga. Da mesma forma, os pontos de verificação de umidade e irrigação podem ser vários, em virtude de um solo mais heterogêneo ou de aplicações diferentes, como diferentes tipos de plantio.

Cada módulo de controle de bomba deverá ter um sensor de nível, do tipo boia eletrônica, digital, que indica se a caixa ou o poço estão vazios, para evitar acionamentos sem carga. Também deverá ter um módulo relé, para ligação do motor da bomba. Ambos os módulos são digitais, sendo a boia eletrônica uma entrada e o módulo relé uma saída para o microcontrolador.

Cada módulo de controle da irrigação contará com um sensor de umidade do solo, além de um módulo relé, para ligar uma válvula solenoide. O sensor de umidade de solo será percebido pelo microcontrolador em sua

interface analógica, indicando um percentual de umidade do solo. As configurações de umidade máxima e mínima devem ser percebidas pelo microcontrolador que aguarda até a umidade mínima para ligar a solenoide e promover a irrigação. Após atingir a umidade máxima desejada, esta será desligada.

Os módulos vão se comunicar entre si usando o protocolo MQTT e o broker da *shiftr.io*. Esse broker gratuito e fácil de configurar vai intermediar as mensagens entre os módulos, bem como possibilitará, através de interfaces específicas, a configuração dos parâmetros mínimos e máximos de cada módulo de irrigação e visualizar o estado de cada elemento do sistema, identificando qual está acionado e a situação identificada pelos sensores.

Dessa forma, estão previstos os seguintes elementos no sistema: módulo de controle de irrigação, módulo de controle da bomba e interface externa (inicialmente web). A tabela 11.1 aponta as mensagens enviadase recebidas e as ações tomadas por cada um dos elementos.

Tabela 11.1 – Mensagens trocadas pela aplicação e suas ações

Elemento	Mensagens enviadas	Mensagens recebidas	Ações tomadas
Módulo Controle Irrigação	1. Umidade do solo percebida 2. Solenoide ligada 3. Solenoide desligada	1. Configuração máxima e mínima da umidade do solo 2. Nível de água	1. Ligar a solenoide quando a umidade estiver abaixo da mínima 2. Desligar a solenoide quando a umidade estiver acima da máxima ou quando faltar água
Módulo Controle Bomba	1. Nível da água 2. Bomba acionada	1. Solenoide ligada 2. Solenoide desligada	1. Ligar a bomba quando houver água e a solenoide estiver ligada 2. Desligar a bomba quando faltar água ou a solenoide estiver desligada
Interface	1. Configuração máxima e mínima da umidade do solo	1. Umidade do solo percebida 2. Solenoide ligada 3. Solenoide desligada 4. Nível da água 5. Bomba acionada	

A interface também pode ser feita por um aplicativo para smartphone. Há dezenas de aplicativos que suportam o protocolo MQTT e apresentam as informações em painéis e gráficos, como pode ser visto na figura 11.2. Essa figura é uma das possíveis telas do aplicativo *MQTT Dashboard*, que possibilita assinar, visualizar, gerar gráficos e publicar informações com o protocolo MQTT.

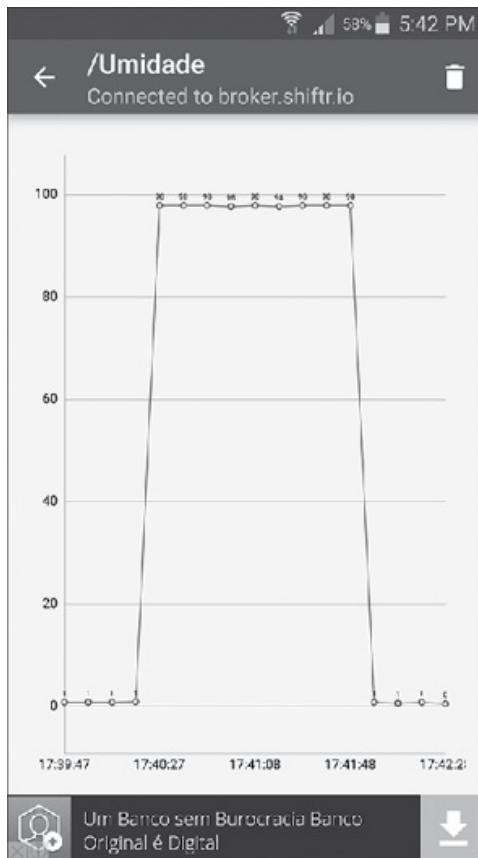



Figura 11.2 – Tela do aplicativo para monitoramento MQTT.

11.3 Módulo irrigação

O módulo irrigação será responsável por medir a umidade do solo, publicá-la e, quando possível e necessário, ligar a solenoide, que possibilitará a irrigação na região de interesse. Só será possível ligar a solenoide quando for identificado o nível alto de água pelo módulo bomba.

O código segue, abaixo. A parte relativa à conexão foi incluída em uma função chamada `connect()`, visto que é possível ocorrer repetidas quedas de conexão, e essa função deve ser chamada constantemente, o que é feito na função `loop()`, caso não seja verificada a conexão.

Na função `setup()`, as entradas e saídas são configuradas, a conexão com o broker é estabelecida com o cliente específico, incluído por meio da biblioteca `MQTTClient` e da declaração da variável `client` do tipo `MQTTClient`. Essa biblioteca está disponível pela pesquisa tanto na IDE Arduino quanto na IDE Eclipse. Também nessa função, são assinados os tópicos de interesse para esse módulo, que incluem a umidade máxima e mínima, caso precisem ser redefinidos, e o nível de água, pois o sistema só será acionado se o nível de água for satisfatório.

Na função `loop()`, inicia-se verificando a conexão para, em seguida, de acordo com um temporizador de 10 s, verificar a umidade do solo, publicar para o broker e verificar as condições para que a solenoide seja ligada. As condições incluem o nível alto da água, além da umidade do solo abaixo da mínima, para ligar; e acima da máxima, para desligar. Definir as condições de umidade máxima e mínima são importantes para garantir uma faixa na qual o sistema permanece ligado ou desligado, para evitar um constante liga-e-desliga, que ocorreria em caso de manter apenas uma umidade de referência.

A função `messageReceived()` é uma função do tipo *callback*, ou seja, ela é chamada em resposta a eventos. Ela já é associada a essa denominação pela biblioteca. Caso necessário, ela pode ser redefinida. Nesse caso, o evento é o recebimento de uma mensagem para um dos tópicos assinados. Os tópicos são assinados na função `setup()`, usando a função `subscribe()`.

```
#include <ESP8266WiFi.h>
#include <MQTTclient.h>
#define SOLENOIDE D0
WiFiClient net;
MQTTClient cliente;
unsigned long lastMillis = 0;
double umidadeMaxima = 60.0, umidadeMinima = 40.0;
bool solenoideLigada = false, caixaCheia = false;
void conectar(); // Predefinição de conectar() que é usado em setup()
void setup() {
    Serial.begin(115200);
    WiFi.begin("ssid", "senha");
    pinMode(D0, OUTPUT);
    cliente.begin("broker.shiftr.io", net);
```

```
    conectar();
}
void conectar() {
    Serial.print("Verificando wifi...");
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(1000);
    }
    Serial.print("\nConectando...");
    while (!cliente.connect("esp8266", "cap_iot_", "iotiot256")) {
        Serial.print(".");
        delay(1000);
    }
    Serial.println("\Conectado!");
    cliente.subscribe("/UmidadeMaxima");
    cliente.subscribe("/UmidadeMinima");
    cliente.subscribe("/NivelAgua");
}

void loop() {
    cliente.loop();
    delay(10); // Evita instabilidade do WiFi
    if(!cliente.connected()) {
        conectar();
    }
    // Publica uma mensagem a cada segundo
    if(millis() - lastMillis > 1000) {
        lastMillis = millis();
        double umidade = double(analogRead(A0))/10.23;
        cliente.publish("/Umidade", String(umidade, 2));
        if (solenoideLigada) {
            if (umidade > umidadeMaxima) {
                digitalWrite(D0, LOW);
                cliente.publish("/Solenoide", "Desliga");
                solenoideLigada = false;
            }
        } else {
            if (umidade < umidadeMinima && caixaCheia) {
                digitalWrite(D0, HIGH);
```

```
                cliente.publish("/Solenoide", "Liga");
                solenoideLigada = true;
            }
        }
    }
}

void messageReceived(String topic, String payload, char * bytes,
        unsigned int length) {
    Serial.print("Mensagem recebida: ");
    Serial.print(topic);
    Serial.print(" - ");
    Serial.print(payload);
    Serial.println();
    if (topic == "/UmidadeMaxima")
        umidadeMaxima = payload.toFloat();
    if (topic == "/UmidadeMinima")
        umidadeMinima = payload.toFloat();
    if (topic == "/NivelAgua") {
        if (payload == "Baixo") {
            caixaCheia = false;
            digitalWrite(D0, LOW);
            cliente.publish("/Solenoide", "Desliga");
            solenoideLigada = false;
        } else if (payload == "Alto")
            caixaCheia = true;
    }
}
```

Dessa forma, usando as funções `publish()` e `subscribe()`, o módulo irrigação é, ao mesmo tempo, *subscriber* e *publisher* para o protocolo MQTT. Ele publica e assina as mensagens.

11.4 Módulo bomba

O módulo de controle da bomba tem funcionamento bem parecido com o do módulo de irrigação. Alguns tópicos MQTT são assinados, outros, publicados. O nível de água é verificado e, caso a solenoide seja acionada, a bomba também será. Elas vão entrar em conjunto para irrigar o sistema. A

bomba é essencial se o reservatório de água está em nível inferior ao nível do jardim a ser irrigado. Do contrário, pode ser descartada. A solenoide, por sua vez, será opcional se a bomba estiver presente, mas é interessante que a mantenha, de qualquer forma, para evitar que entre água no encanamento, o que dificulta o seu funcionamento.

A presença da bomba e da solenoide foi considerada, de qualquer forma. Assim, as mensagens publicadas por esse módulo são: nível de água e bomba, ligada ou desligada. E o módulo assina o tópico da solenoide, enviado pelo módulo irrigação. Assim que a solenoide for ativada, o módulo ligará a bomba.

```
#include <ESP8266WiFi.h>
#include <MQTTClient.h>
#define BOIA D0
#define BOMBA D1
WiFiClient net;
MQTTClient cliente;
unsigned long lastMillis = 0;
bool solenoideLigada = false, bombaLigada = false;
void conectar() {
    Serial.print("Verificando wifi...");
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(1000);
    }
    Serial.print("\nConectando...");
    while (!cliente.connect("esp8266", "cap_iot_", "iotiot256")) {
        Serial.print(".");
        delay(1000);
    }
    Serial.println("Conectado!");
    cliente.subscribe("/Solenoide");
}

void setup() {
    Serial.begin(115200);
    WiFi.begin("ssid", "senha");
    pinMode(BOMBA, OUTPUT);
```

```
    pinMode(BOIA, INPUT);
    cliente.begin("broker.shiftr.io", net);
    conectar();
}

void loop() {
    cliente.loop();
    delay(10); // Tempo necessário para evitar algumas instabilidades
    if(!cliente.connected()) {
        conectar();
    }
    // Publica uma mensagem periodicamente a cada segundo
    if(millis() - lastMillis > 1000) {
        lastMillis = millis();
        int nivel = digitalRead(BOIA);
        if (nivel == LOW)
            cliente.publish("/NivelAgua", "Baixo");
        else if (nivel == HIGH)
            cliente.publish("/NivelAgua", "Alto");
        if (bombaLigada && nivel == LOW) {
            digitalWrite(BOMBA, LOW);
            cliente.publish("/Bomba", "Desliga");
            bombaLigada = false;
        } else if (solenoideLigada && nivel == HIGH && !bombaLigada) {
            digitalWrite(BOMBA, HIGH);
            cliente.publish("/Bomba", "Liga");
            bombaLigada = true;
        }
    }
}

// Função requerida e chamada pela biblioteca MQTT, exatamente nesse
formato
void messageReceived(String topic, String payload, char * bytes,
        unsigned int length) {
    Serial.print("Mensagem recebida: ");
    Serial.print(topic);
    Serial.print(" - ");
    Serial.print(payload);
```

```
    Serial.println();
    if (topic == "/Solenoide") {
        if (payload == "Liga")
            solenoideLigada = true;
        else if (payload == "Desliga")
        solenoideLigada = false;
    }
}
```

O funcionamento dos módulos irrigação e bomba são independentes de interfaces e visualizações. O protocolo MQTT cumpre sua função, de comunicação M2M, ou seja, *machine-to-machine*, ou máquina a máquina, sem a necessidade de intervenção humana. As interfaces a ser usadas para acesso aos dados do sistema são apenas para acompanhamento e configuração.

11.5 Interface web

Como dito anteriormente, é possível acessar os dados e as configurações desse sistema por meio de aplicativos para smartphone. Assim, essa interface web é dispensável. No entanto, como pode ser útil para esse tipo de sistema, será apresentada como mais uma contribuição na comunicação web com IoT.

Essa interface foi escolhida para ser desenvolvida em JavaScript, em função da disponibilidade de biblioteca para essa linguagem, bem documentada para o broker utilizado (shiftr.io).

A linguagem JavaScript é usada para desenvolver aplicações que executam no cliente web, ou seja, no navegador do usuário, seja Internet Explorer, Google Chrome ou Firefox. A princípio, não depende do servidor e poderia estar localizada no computador do usuário.

O código JavaScript fica, normalmente, embutido em uma página HTML, podendo usar um ou mais arquivos de biblioteca. Nesse caso, optou-se por um arquivo com o código principal em JavaScript, além das inserções no código HTML.

O código HTML inclui as bibliotecas do broker e o código específico para essa aplicação que está salvo em *script.js*. E define três identificadores: `max`,

`min`, `button` e `myList`. Os dois primeiros serão usados para inserir os novos limites máximo e mínimo da umidade a ser configurada. O botão será usado para submeter os valores ao broker, e o campo `myList` será usado para apresentar todas as mensagens recebidas como assinante. O código HTML é este:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="https://code.jquery.com/jquery-2.2.3.min.js">
</script>
        <script src="https://assets.shiftr.io/js/mqtt-latest.js">
</script>
        <script src="script.js" charset="utf-8"></script>
    </head>
    <body>
        M&iacute;nimo: <input type=text id="min">    M&aacute;ximo:
            <input type=text id="max">
        <button id="button">Enviar</button>
        <ul id="myList"> </ul>
    </body>
</html>
```

O código JavaScript segue a mesma lógica do código para o microcontrolador. Assinam-se alguns tópicos e trata-se seu recebimento com uma função que temo procedimento similar ao da função *callback*. O evento *click* do botão é tratado publicando-se os limites máximo e mínimo da umidade que serão recebidos pelo módulo irrigação.

A função para tratamento das mensagens recebidas apenas as insere em uma lista para que apareçam no campo `myList` definido no código HTML. Para isso é necessário criar um elemento de texto, ligá-lo a um elemento de lista e inseri-lo na lista.

A conexão é feita por meio da função `connect()`. A assinatura, por meio da função `subscribe()`. E a publicação, pela função `publish()`. Essas funções estão definidas na biblioteca do broker inserida no código HTML. A seguir é apresentado o código JavaScript salvo no arquivo *script.js*.

```
$(function() {
    var client =
mqtt.connect('mqtt://cap_iot_:iotiot256@broker.shiftr.io', {
        clientId: 'javascript'
    });

    client.on('connect', function() {
        console.log('Cliente conectado!');
        client.subscribe('/Umidade');
        client.subscribe('/Solenoide');
        client.subscribe('/Bomba');
        client.subscribe('/NivelAgua');
    });

    client.on('message', function(topic, message) {
        var msg = message.toString();
        console.log('Nova mensagem', topic, msg);
        var node = document.createElement("LI");
        var textnode = document.createTextNode(topic.concat(": ",
msg));
        node.appendChild(textnode);
        document.getElementById("myList").appendChild(node);
    });

    $('#button').click(function() {
        client.publish('/UmidadeMinima',
document.getElementById("min").value);
        client.publish('/UmidadeMaxima',
document.getElementById("max").value);
        window.alert("Valores enviados");
    })
})
```

A interface fica como mostra a figura 11.3.

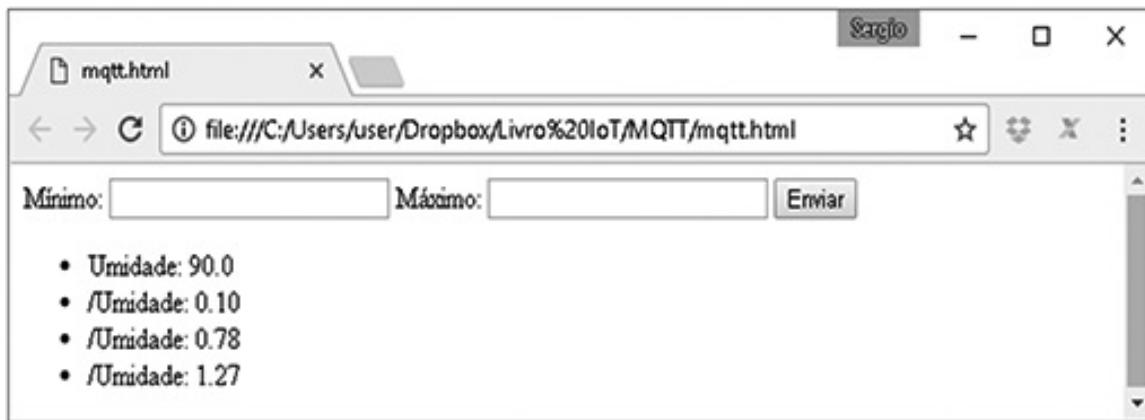



Figura 11.3 – Interface web para monitoramento da umidade do solo.

A umidade percebida pelo sensor do módulo de irrigação e o nível de água foram configurados para ser enviados a cada 10 segundos. Numa aplicação prática, esse valor poderia aumentar para alguns minutos, visto que não há necessidade de monitoramento com tal frequência.

CAPÍTULO 12

Controle de acesso web utilizando RFID

O controle de acesso a ambientes é uma das aplicações típicas de Internet das Coisas. Registrar quem entra e sai, além de ter uma forma fácil de controlar o acesso, é o objetivo da aplicação apresentada neste capítulo. Um módulo com o microcontrolador ESP8266, usando um módulo leitor de RFID, é usado para identificar e liberar o acesso feito por uma fechadura elétrica ligada a uma das portas GPIO. O cadastro, bem como a emissão de relatórios, é feito pela web e registrado em um banco de dados. O banco de dados MySQL e a interface web de acesso desenvolvida na linguagem PHP são apresentados neste exemplo.

12.1 Apresentação

O controle de acesso a ambientes restritos usando RFID é uma das aplicações mais comuns de IoT. Além de substituir a fechadura com várias vantagens em relação à segurança, também pode registrar os horários de entrada e saída de cada pessoa autorizada.

O registro dos acessos deve ser feito em um banco de dados para garantir a confiabilidade dos dados, além de facilitar seu acesso e gerenciamento. Um banco de dados local resolveria o problema, visto que existe biblioteca Arduino para fazer acesso direto ao banco de dados MySQL. No entanto um banco de dados local exige infraestrutura computacional, manutenção e backup. Manter esse banco de dados na nuvem é uma solução bem mais barata, além de facilitar seu acesso remoto.

O acesso direto a servidores de banco de dados pela internet não tem sido visto como seguro em consequência da forma de associação de usuários e senhas à manutenção do banco de dados. Logo, é interessante trabalhar em outra proposta de interface. Para este trabalho, a interface para o banco de dados será feita via web, garantindo bom desempenho e segurança. A

linguagem PHP foi escolhida pela sua simplicidade e rapidez no desenvolvimento. A solução será hospedada em um servidor de hospedagem gratuita – *hostinger.com.br* –, que ficará responsável pelo banco de dados e pela hospedagem web, sendo gratuito dentro de alguns limites de recursos e capacidades.

Finalmente, um módulo com o ESP8266 ligado a um leitor de RFID fará a leitura dos cartões, permitindo ou não o acesso por meio de uma fechadura eletrônica e registrando todo acesso.

12.2 Casos de uso

O sistema terá, basicamente, quatro casos de uso:

1. acesso-padrão, para abertura da porta;
2. inserção de uma nova tag;
3. remoção de uma tag;
4. listagem de todos os acessos feitos por um determinado período.

O acesso-padrão será iniciado pela aproximação da tag RFID do leitor, que estará ligada ao ESP8266. Caso a tag esteja previamente cadastrada no banco de dados, o acesso será liberado pelo acionamento de um relé ligado a uma porta GPIO. O relé comanda o acionamento da abertura da porta. Se a tag utilizada não estiver cadastrada no banco de dados, a porta não será aberta.

A inserção de uma nova tag envolve a aproximação da tag não cadastrada ao leitor, e a inserção dos dados pela interface web. Por meio de uma URL previamente conhecida é possível iniciar o cadastro. A URL solicitará a aproximação da tag do leitor. Assim que o sistema identificar a nova tag, vai apresentar a opção de inserção de nome e documento de identificação. Outros dados poderiam ser armazenados nesse passo. Após a inserção, a tag deve ser reconhecida para liberar o acesso à porta.

A remoção de uma tag é um processo totalmente web, envolvendo apenas a interação com o banco de dados. A partir da remoção de uma tag, ela não deve mais ser reconhecida pelo leitor RFID.

A listagem dos acessos para um determinado período também é um

processo que só envolve a interface web e o banco de dados. O usuário deve entrar no período inicial e no período final para o qual quer realizar a consulta, e o sistema deve gerar uma listagem com todos os acessos feitos nesse período.

12.3 Banco de dados

A definição do banco de dados se inicia pela construção do diagrama entidade-relacionamento (ER). Nossa aplicação terá duas entidades: Usuário e Acesso. O usuário deve conter as informações imprescindíveis sobre os usuários do sistema, incluindo o código da tag RFID e os seus dados pessoais. O acesso constará apenas do registro de cada acesso, indicando a tag RFID usada para o acesso e a data e a hora em que o acesso ocorreu.

A figura 12.1 mostra o diagrama entidade-relacionamento do banco de dados usado.

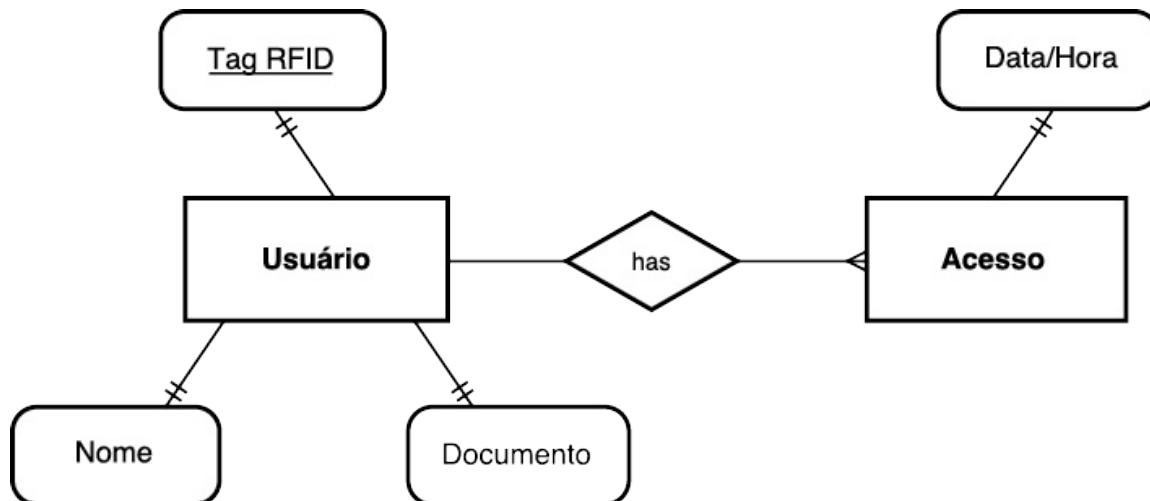



Figura 12.1 – Modelagem do banco de dados.

A partir do diagrama entidade-relacionamento foi criado o modelo relacional. O banco de dados usado tem duas tabelas; cada uma representa uma das entidades do diagrama. Como há um relacionamento um-para-muitos entre a tabela `Usuario` e a tabela `Acesso`, visto que um usuário faz vários acessos, a tabela `Acesso` herda a chave da tabela `Usuarios`, que é a tag RFID.

`Usuarios`(Tag, Nome, Documento)

`Acessos`(Tag, DataHora)

É necessário, ainda, definir as chaves primárias das tabelas. Como as chaves devem respeitar o requisito de unicidade, as tags são boas candidatas à chave na tabela `Usuarios`. Na tabela `Acessos`, no entanto, não existe nenhuma informação com unicidade, visto que as tags podem aparecer várias vezes nos diversos acessos com a tag, e em uma mesma data e mesma hora é possível fazer dois acessos com tags diferentes. Assim, a solução é criar um novo campo numérico, autoincrementável, para servir de chave primária. O esquema relacional, com as chaves e os tipos, fica assim:

- **Usuarios** (Tag: texto(20), chave primária;

  Nome: texto(80),

  Documento: texto(20))

- **Acessos** (Num: inteiro autoincrementável, chave primária;

  Tag: texto(20), chave estrangeira da tabela Usuários;

  DataHora: Tempo);

As tabelas são criadas no banco de dados usando código SQL. O comando para a criação das tabelas é o `CREATE TABLE`. A sintaxe para a criação das tabelas seria como se vê a seguir:

```
CREATE TABLE Usuarios (
    Tag varchar(20) primary key,
    Nome varchar(80),
    Documento varchar(20)
);

CREATE TABLE Acessos (
    num integer primary key auto_increment,
    Tag varchar(20),
    foreign key (Tag) References Usuarios(Tag) ,
    recorded timestamp
);
```

Além dessas duas tabelas, uma tabela para registrar os acessos não autorizados também foi criada. Essa tabela tem por objetivo maior possibilitar a inserção de novas tags. A ideia é que, sempre que um novo

cadastro for feito, ele deve se iniciar por uma leitura da tag ainda não cadastrada. Após a leitura, por um período de tempo predefinido, e possivelmente configurável, o sistema vai aguardar a inserção de um novo usuário usando essa tag. A tabela de acesso não autorizado armazena apenas o código RFID da tag e a hora em que a leitura foi feita.

O banco de dados foi criado no servidor de hospedagem hostinger.com.br. O servidor oferece hospedagem gratuita de até dois bancos de dados, além de interface *PHPMyAdmin* para acesso ao banco.

12.4 Interface web

A interface web desta aplicação será feita em PHP e também será hospedada no Hostinger. Serão três arquivos para o acesso, a saber:

1. acesso para inserção de novo usuário;
2. acesso para registro de acesso;
3. acesso para leitura das tags *RFID* cadastradas.

A inserção de novo usuário se inicia pela interface web. Ao acessar o site, o usuário digita o nome e o documento do usuário e aproxima a tag a ser cadastrada do leitor RFID. Como a tag é desconhecida do sistema, a porta não é aberta, mas o microcontrolador envia seu código para a interface de inserção. Caso o processo de cadastro esteja em andamento, a tag e os dados serão registrados no banco de dados.

O registro de acesso é uma interface simples, sem formulário ou página. O microcontrolador usa o método POST para enviar a tag identificada para o registro. Um arquivo com código PHP recebe o código da tag e registra o acesso no banco de dados.

O acesso para a leitura das tags RFID cadastradas é uma simples consulta ao banco de dados, de todas as tags cadastradas. Essas tags são enviadas para o microcontrolador e salvas para validação dos usuários. São mantidas na memória do microcontrolador, que pode validar o acesso mesmo que algum problema de conexão impeça seu acesso à interface web. Essas informações devem ser renovadas periodicamente por questões de segurança.

12.4.1 Acesso RFID

Ao aproximar uma tag do leitor RFID, o código em execução no ESP8266 deve acessar a interface web para identificar se a tag está devidamente cadastrada. Caso não esteja cadastrada, será possível dar início a um processo de cadastramento de um novo usuário, como será apresentado na próxima seção.

O programa em PHP é apresentado a seguir. Ele inicia com uma consulta para verificar se a tag lida está cadastrada no banco de dados. Caso esteja, será registrado um acesso na tabela `Acessos`. Do contrário, ela será registrada na tabela `Insercao`, que vai disponibilizar essa informação para o programa de novas inserções, que será apresentado na próxima seção. Esse dado só será usado se uma nova inserção for realizada nos cinco minutos seguintes, podendo esse tempo ser ajustado no código. O arquivo foi chamado de *consult.php*.

```
<?php
    // Cria a conexão
    $conn = new mysqli("servidor", "usuario", "senha", "banco");
    // Verifica se foi criada com sucesso
    if ($conn->connect_error) {
        die("Falha na conexão: " . $conn->connect_error);
    }

    $sql = "SELECT * FROM Usuarios WHERE tag = '" . $_POST[tag] . "'";
    $result = $conn->query($sql);
    if ((!$result) || ($result->num_rows == 0)) { // Tag não cadastrada
        echo "NOTAG\n";
        $result->free();
        $sql = "INSERT INTO Insercao (TAG) VALUES ('". $_POST[tag] .
"')";
        $result = $conn->query($sql);
    } else { // Tag existente, registra acesso
        echo "OK\n";
        $result->free();
        $sql = "INSERT INTO Acessos (TAG) VALUES ('". $_POST[tag] .
"')";
        $result = $conn->query($sql);
```

```
    }
    $conn->close();
?>
```

O método para envio da tag do ESP8266 para o programa em PHP é uma requisição HTTP do tipo POST. Essa requisição encapsula os dados, que são recebidos no vetor `$_POST` no programa PHP. O programa envia como resposta a palavra *OK*, caso a tag esteja cadastrada, ou *NOTAG*, caso não seja localizada na tabela.

12.4.2 Inserção de usuário

A inserção de uma nova tag, que pode estar associada a um novo usuário ou até a um usuário já existente, portador de outra tag, se dá em três etapas. A primeira etapa é a leitura da tag a ser cadastrada. Na segunda etapa, um formulário é apresentado ao usuário para cadastro do nome documento. E, por último, esses dados são inseridos no banco de dados. Dessa forma, o programa em PHP, listado a seguir, é executado pelo menos três vezes para fazer uma inserção. A cada etapa, parte da inserção é concluída. O arquivo foi chamado *insere.php*.

```
<HTML><BODY>
<?php
    // Cria a conexão
    $conn = new mysqli("servername", "username", "password", "dbname");
    // Verifica se foi criada com sucesso
    if ($conn->connect_error) {
        die("Falha na conexão: " . $conn->connect_error);
    }

    if (!$_POST[Nome]) { // 1a Etapa: Se não tem os dados, verifica se
uma nova
                        // tag foi lida...

        $sql = "SELECT * FROM Insercao
            WHERE DataHora > NOW() - INTERVAL 5 MINUTE
            ORDER BY DataHora DESC LIMIT 1";
        // Recupera a última tag enviada
        $result = $conn->query($sql);
```

```
        if ((!$result) || ($result->num_rows == 0)) {
            echo "Não foi identificada nenhuma tag. Aproxime uma tag do
leitor <BR>";
            header("Refresh:3"); // Tag não lida... Recarrega a página
a cada 3s
                                 // até a leitura.
        } else { // 2a Etapa: Tag lida, monta o formulário
            $row = $result->fetch_assoc();
            echo "<form action=insere.php method=POST>
            <p>Nome: <input type=text name=Nome /></p>
            <p>Documento: <input type=text name=Documento /></p>
            <p>Tag: <input type=text name=Tag value=" . $row['TAG'] . "
/></p>
            <p><input type=submit /></p>
            </form>";
        }

    } else { // 3a Etapa: Se os dados já foram enviados, insere no
banco
        $sql = "INSERT INTO Usuarios VALUES('".$_POST['Tag']."','
            ".$_POST[Nome]."','".$_POST[Documento]."')";
        $result = $conn->query($sql);
        if ($result)
            echo "Inserção realizada com sucesso<BR>";
    }
?>
</body></html>
```

A primeira etapa, sinalizada no código pelo comentário com referência à **1ª etapa**, será executada quando o programa não receber nenhum parâmetro. Os parâmetros são recebidos no vetor `$_POST`, que é verificado na primeira linha do programa. Se os parâmetros não são encontrados, o programa já executa a primeira etapa.

Na primeira etapa, o programa pesquisa a tabela `Insercao` no banco de dados, à procura de uma tag inserida nos últimos cinco minutos. Se não encontrar nenhuma tag inserida nos últimos cinco minutos, a página será recarregada até que uma nova tag seja aproximada do leitor. O código para essa consulta ao banco de dados é:

```
SELECT * FROM Insercao
      WHERE DataHora > NOW() - INTERVAL 5 MINUTE
      ORDER BY DataHora DESC LIMIT 1
```

Os dados da tabela `Insercao` são selecionados pelo campo `DataHora`, considerando o momento atual `NOW()`, menos o intervalo de cinco minutos. O resultado é ordenado pelo campo `DataHora` de forma decrescente (`DESC`), para que os dados mais recentes sejam apresentados antes. E são limitados a apenas um resultado (`LIMIT 1`), o que garante que será recuperada apenas última tag nova aproximada do leitor. A figura 12.2 mostra a mensagem enviada ao usuário até que uma tag seja aproximada do leitor.

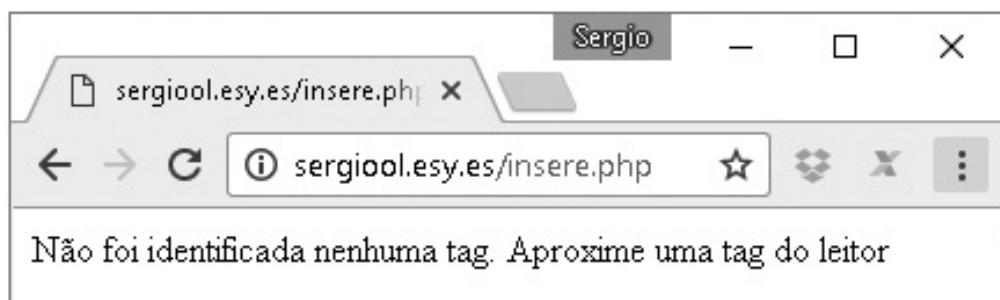



Figura 12.2 – Interface web para início do cadastro.

Assim que a consulta devolve um resultado, indicando que uma tag foi aproximada do leitor, o programa segue para a segunda etapa, na qual o formulário é montado. Ele contém os campos para a entrada de dados do usuário, incluindo `Nome`, `Documento` e `Tag`, sendo essa já preenchida com o último valor lido, não devendo ser alterada pelo usuário. Ao final dessa fase, o usuário terá um formulário para inserir os dados. A figura 12.3 mostra o formulário gerado.

Figura 12.3 – Interface web para cadastro da tag.

A terceira etapa é iniciada assim que os dados são preenchidos e enviados, pelo botão *Enviar* do formulário. O programa é novamente executado, agora com os dados disponíveis no vetor `$_POST`. Uma inserção é feita utilizando-se esses dados. Caso a inserção seja realizada corretamente, a mensagem da figura 12.4 será exibida.

Figura 12.4 – Interface web de conclusão do cadastro.

A partir do momento em que a inserção é realizada, a tag deve estar habilitada para abrir o acesso, sendo aceita pelo leitor.

12.4.3 Apagar usuários e tags

A próxima interface a ser apresentada é a remoção de usuários e tags. Essa interface tem acesso apenas ao banco de dados. O programa lê todos os registros de usuários e tags da tabela `Usuarios` e os apresenta à tela. Os registros a ser removidos devem ser marcados, e, após apertar o botão enviar, o programa identifica quais registros foram marcados e os exclui do banco de dados com o comando `DELETE`.

Observe que, no programa, o comando apagar (`DELETE`) aparece antes da consulta (`SELECT`). Isso ocorre porque, se o usuário enviar o comando de apagar, os registros devem ser apagados antes que sejam novamente apresentados, possibilitando excluir outros registros. Caso nada tenha sido, ainda, selecionado para ser apagado, o comando `DELETE` não tem efeito. O arquivo foi chamado *remove.php*.

```
<html><body>
<?php
    // Cria a conexão
    $conn = new mysqli("servername", "username", "password", "dbname");
    // Verifica se foi criada com sucesso
```

```
    if ($conn->connect_error) {
        die("Falha na conexão: " . $conn->connect_error);
    }
    foreach($_POST as $val) {
        $sql = "DELETE FROM Usuarios WHERE TAG='" . $val . "'";
        $result = $conn->query($sql);
        if ($result) echo "Tag " . $val . " removida com sucesso.
<BR>";
    }
?>

Selecione as tags a serem excluídas:
<form action=remove.php method=post>

<?php
    $sql = "SELECT TAG,Nome FROM Usuarios";
    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
        // Imprime os dados de cada linha da tabela
        while($row = $result->fetch_assoc())
            echo "<input type=checkbox name=check ".$row["TAG"]."
value= ".
                \ $row["TAG"].">".$row["TAG"]."-". $row["Nome"]."
<br>";
    } else {
        echo "Resultado vazio";
    }
    $conn->close();
?>
<input type="submit" value="Submit">
</form>
</html></body>
```

A tela para a remoção dos usuários é mostrada na figura 12.5.

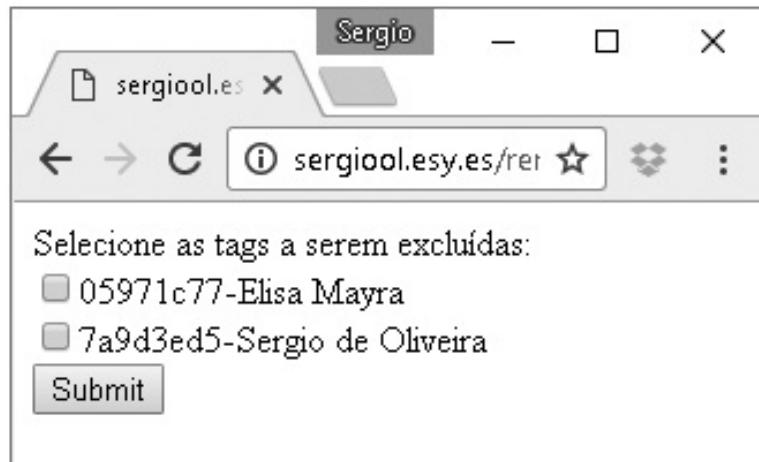



Figura 12.5 – Interface web para a remoção de tags.

Após a seleção dos registros e o envio, estes serão apagados, e não será possível recuperá-los.

12.4.4 Listar acessos

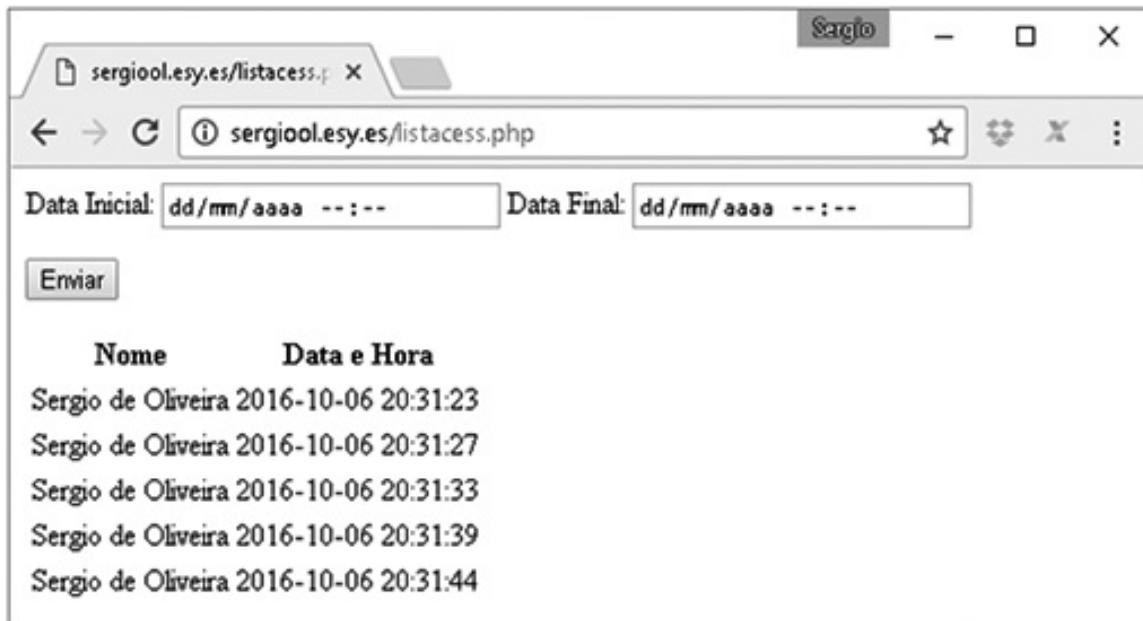
A página para listar acessos também só tem interface com o banco de dados. Dois campos recebem as datas e as hora iniciais e os finais para a consulta. E um botão, assim que acionado, apresenta todos os acessos feitos no período, indicando a tag e o nome associado a ela.

A consulta SQL usada, nesse caso, envolve as duas tabelas: `Acessos` e `Usuarios`. O cruzamento entre as tabelas é necessário para que os nomes dos usuários portadores das tags sejam recuperados. A consulta SQL para essa finalidade é:

```
SELECT Usuarios.Nome, Acessos.DataHora
FROM Usuarios, Acessos
WHERE Acessos.DataHora > 'DataInicial' AND
      Acessos.DataHora < 'DataFinal' AND
      Acessos.tag = Usuarios.TAG
```

A consulta deve ser restrita para que só sejam exibidos os registros que confrontam o campo tag da tabela `Acessos` com o campo tag da tabela `Usuarios`, por isso, a cláusula `Acessos.tag = Usuarios.TAG` foi incluída na consulta.

Veja o código a seguir. O arquivo foi chamado *listacess.php*.

```
<html><body>
```

```
    <form action=listacess.php method=POST>
    <p>Data Inicial: <input type=datetime-local name=DataI />
       Data Final: <input type=datetime-local name=DataF /></p>
    <p><input type=submit /></p>
    </form>
<?php
    if ($_POST['DataI']) {
        $conn = new mysqli("servername", "username", "password",
"dbname");
        // Verifica conexão
        if ($conn->connect_error) {
            die("Falha na conexão: " . $conn->connect_error);
        }
        $sql = "SELECT Usuarios.Nome, Acessos.DataHora
                FROM Usuarios, Acessos
                WHERE Acessos.DataHora > '" . $_POST['DataI'] . "' AND
                Acessos.DataHora < '" . $_POST['DataF'] . "' AND
                Acessos.tag = Usuarios.TAG";
        echo $sql;
        $result = $conn->query($sql);
        if ($result->num_rows > 0) {
            // Imprime os dados de cada linha da tabela
            echo "<TABLE><tr><th>Nome<th>Data e Hora";
            while($row = $result->fetch_assoc()) {
                echo "<tr><td>" . $row["Nome"] . "<td>" .
$row["DataHora"] . "<BR>";
            }
        } else {
            echo "Resultado Vazio";
        }
        $conn->close();
     }
?>
</html></body>
```

O resultado dessa execução gera uma página, como pode ser visto na figura 12.6. A página inicialmente não apresenta listagem alguma, apenas os campos para inserção de datas e horários iniciais e finais. Após completar esses campos e enviar, uma listagem é apresentada logo a seguir.

Figura 12.6 – Interface web para listar acessos.

Seria possível criar vários tipos de relatórios como esse, bem como filtrar por períodos específicos de tempo como: na última hora, no dia atual, na última semana etc. A elaboração desses relatórios e o aprendizado da linguagem PHP, no entanto, não são o objetivo principal deste livro.

12.5 Dispositivo IoT

Após o desenvolvimento da interface web, será apresentado o código para o controle do leitor RFID, que será executado no módulo ESP8266. A interface web será acessada tanto pelo dispositivo IoT quanto pelos usuários, para realizar a gestão do sistema. A integração desses módulos é fundamental para o funcionamento do sistema.

O dispositivo IoT será composto de um módulo com o microcontrolador, um leitor RFID MFRC522, com interface serial SPI; e um módulo relé de um canal que pode ser ligado à uma fechadura eletrônica ou uma catraca, responsável pelo acesso que se queira controlar. As tags de acesso podem ser cartões, chaveiros ou qualquer tipo de tag RFID na frequência de 13,56 MHz aceita por esse módulo. Substituindo o leitor RFID pelo módulo PN532, seria possível receber sinal NFC, disponível em smartphones, em substituição à tag RFID.

A ligação do leitor RFID MFRC522 é feita através da interface SPI do

ESP8266. Para isso devem-se usar os pinos correspondentes do módulo. Para a versão do módulo ESP12, também conhecida como NodeMCU v0.9, os seguintes pinos devem ser interligados:

- VCC – qualquer pino de 3,3 volts;
- GND – qualquer pino GND do módulo;
- MISO – GPIO12 ou D6;
- MOSI – GPIO13 ou D7;
- SCK (ou CLK, Clock) – GPIO14 ou D5;
- SDA (o mesmo que CS ou SS, seleção de escravo) – qualquer pino GPIO; para o exemplo, foi escolhido o pino GPIO04 ou D2;
- RST – qualquer pino GPIO; para o exemplo, foi escolhido o pino GPIO02 ou D4.

Os pinos de SDA e SCK escolhidos são associados em software, na inicialização da placa, pelo construtor `mfrc522(SDA_PIN, RST_PIN)`. O módulo relé está ligado à porta GPIO16 ou ao pino D0.

O código é mostrado a seguir.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <SPI.h>
#include <MFRC522.h>

/* Ligações do MFRC522 para o ESP8266 (ESP-12)
RST  = GPIO2 / SDA(SS) = GPIO4
MOSI = GPIO13 / MISO   = GPIO12
SCK  = GPIO14 / GND    = GND
3.3V = 3.3V */

#define RST_PIN 2  // RST-PIN MFRC522 - RFID - SPI - Modulo GPIO02
#define SS_PIN 4   // SDA-PIN MFRC522 - RFID - SPI - Modulo GPIO4

MFRC522 mfrc522(SS_PIN, RST_PIN); // Cria acesso MFRC522

void setup() {
    Serial.begin(115200);
```

```
    delay(250);
    Serial.println();
    Serial.println("Iniciando....");
    SPI.begin();          // Inicia a serial SPI para o leitor
    mfrc522.PCD_Init();   // Inicia o leitor RFID
    pinMode(D0, OUTPUT);  // Saída que aciona a abertura da porta
    WiFi.begin("ssid", "senha");
    int tentativas = 0;
    while ((WiFi.status() != WL_CONNECTED) && tentativas++ < 20) {
        delay(500);
        Serial.print(".");
    }
    if (WiFi.status() == WL_CONNECTED)
        Serial.println("WiFi conectado");
    }

    char * read_RFID(char *buffer) { // Função para ler as tags
        // Verifica se há cartões presentes e lê um cartão
        if (mfrc522.PICC_IsNewCardPresent() &&
mfrc522.PICC_ReadCardSerial()) {
           buffer = dump_byte_array(mfrc522.uid.uidByte,
mfrc522.uid.size, buffer);
           // Mostra o código da tag em hexadecimal
           Serial.printf("Tag UID:%s\n", buffer);
           return buffer;
        } else
           return NULL;
}
// Função para converter código RFID para char [] em hexadecimal
char * dump_byte_array(byte *buffer, byte bufferSize, char * result) {
    for (byte i = 0; i < bufferSize; i++) {
        char num[3];
        itoa(buffer[i], num, 16);
        if (buffer[i] <= 0xF) strcat(result, "0");
        strcat(result, num);
    }
    return result;
}
```

```
void loop() {
    char code_RFID[20] = "";
    if (read_RFID(code_RFID)) { // Verificar se o cartão foi lido

        if(WiFi.status() == WL_CONNECTED) { // Verifica a conexão Wifi
            HTTPClient http;
            Serial.print("[HTTP] Início...\n");
            // Configura URL, cabeçalhos e monta o método POST com o
código da TAG
            http.begin("http://sergiool.esy.es/consult.php");
            http.addHeader("Content-Type", "application/x-www-form-
urlencoded");
            int httpCode = http.POST(String("tag=") + code_RFID);
            // httpCode será negativo em caso de erro
            if(httpCode > 0) {
            // Obteve resposta do servidor
            Serial.printf("[HTTP] POST... código: %d\n", httpCode);
            // Achou o arquivo
            if(httpCode == HTTP_CODE_OK) { // Conectou, atualiza o
arquivo
                String payload = http.getString();
                Serial.println(payload);
                if (payload == "OK") {
                    digitalWrite(D0, HIGH);
                    delay(100); // Tempo suficiente para acionar o
acesso
                    digitalWrite(D0, LOW);
                }
        }
    } else
        Serial.printf("POST Erro: %s\n",
http.errorToString(httpCode).c_str());
        http.end();
        delay(300); // Tempo suficiente para acionar o acesso
    }
  }
}
```

O código pode ser entendido da seguinte forma: na função `setup()`, as inicializações são feitas, o que inclui a rede WiFi, a porta SPI, o leitor

MFRC522 e a porta na qual será ligado o módulo relé. Sem novidades nessa função.

Uma função para a leitura das tags foi definida. Trata-se da função `read_RFID()`, que verifica se uma tag foi aproximada do leitor. Em caso positivo, retorna o código correspondente no formato `char []`, em base hexadecimal. Esse formato foi escolhido por conta de sua maior facilidade em manipular e registrar dados textuais no banco de dados. A função `dump_byte_array()` foi definida para fazer a conversão do código lido como um vetor de bytes para o vetor de caracteres em hexadecimal. Essa função é chamada logo após a leitura da tag. Caso nenhuma tag seja lida, o valor *NULL* será retornado.

A função `loop()` se inicia com a leitura da tag. Caso não seja realizada, o programa seguirá, aguardando até que a leitura seja feita. Quando esta é concluída, o programa inicia o acesso ao programa em PHP *consult.php*, disponível na web, que faz a interface para consulta da tag no banco de dados. Caso a consulta funcione corretamente e a tag esteja presente no banco de dados, o programa em PHP retornará a palavra “OK”. Se a resposta assim estiver, a porta GPIO ligada ao módulo relé será acionada.

O método para acesso ao programa *consult.php* é o método POST. Ele é mais seguro que o método GET, que seria a alternativa, por encapsular os dados enviados e, possivelmente encriptá-los, caso o protocolo usado seja HTTPS, o que é uma possibilidade para o ESP8266.

12.6 Agregando persistência local

Um problema para aplicações na nuvem, como esta apresentada neste capítulo, é que a falta de conectividade pode comprometer a aplicação. Nem sempre é possível garantir a conectividade, e, na aplicação apresentada, sem conectividade, não seria possível liberar o acesso, pois o código da tag RFID não poderia ser conferido.

Uma alternativa para resolver esse problema seria agregar persistência local à aplicação, mantendo uma cópia do banco de dados, ou de parte dele, em arquivos. Assim, caso a conexão não seja viável, os dados locais serão usados em substituição.

Para a aplicação apresentada, não é necessário manter cópia de todo o

banco de dados para acesso em caso de perda de conectividade. É suficiente manter uma lista dos códigos de acesso válidos, bem como uma lista de todos os acessos autorizados durante o período sem conexão. Isso pode ser feito facilmente usando o sistema de arquivos disponível no ESP8266 e acessado pela interface SPI, conhecida como SPIFFS (SPI Flash Filesystem).

O SPIFFS é um sistema de arquivos mantido na memória flash do ESP8266, que pode chegar a 3 MB, dependendo do modelo da placa, visto que essa memória é uma estrutura externa ao microcontrolador. Uma implementação disponível para o ambiente Arduino tem funções para o acesso ao sistema de arquivos padrão POSIX, similar ao usado no Linux e na linguagem C padrão. Para usar, basta incluir o arquivo *FS.h*, disponível nessa plataforma.

Para agregar persistência local, será necessário manter um arquivo com todos os códigos válidos salvos e verificar periodicamente esse arquivo para ver se ocorreram atualizações, como inclusões ou exclusões. Caso ocorram atualizações, o arquivo deve ser atualizado. Para isso será necessário criar uma nova interface em PHP para recuperar todas as tags válidas.

O código em PHP para fazer essa consulta ao banco é mostrado a seguir. Ele é bem simples e apenas repassa a consulta ao banco. O arquivo foi chamado *listtags.php*.

```
<?php

// Cria a conexão
$conn = new mysqli("servidor", "usuário", "senha", "banco");
// Verifica a conexão
if ($conn->connect_error)
    die("Falha na conexão: " . $conn->connect_error);
// Encerra caso não consiga conectar

$sql = "SELECT tag FROM Usuarios";
$result = $conn->query($sql);
if ($result->num_rows > 0)
    // Imprime cada linha da resposta, que deve conter uma tag
    while($row = $result->fetch_assoc())
        echo $row["tag"] . "\n";
```

```
$conn->close();

?>
```

Para verificar se os dados salvos correspondem aos dados atualizados no banco de dados foi criada uma função que faz acesso ao programa em PHP e compara a resposta com a informação salva. A função é mostrada a seguir e foi inserida no programa do ESP8266.

```
void atualiza_local() {
    HTTPClient http;
    Serial.print("[HTTP] Início...\n");
    // Configura URL
    http.begin("http://sergiool.esy.es/list.php");
    if(http.GET() == HTTP_CODE_OK) { // Conectou, atualiza o arquivo
        String remoto = http.getString();
        Serial.printf("Remoto: %s\n", remoto.c_str());
        File f = SPIFFS.open("/f.txt", "r");
        String local = f.readString();
        Serial.printf("Local: %s\n", local.c_str());
        f.close();
        if (remoto != local) {
            Serial.println("Atualizando ...");
            File f = SPIFFS.open("/f.txt", "w");
            f.print(remoto);
        }
    }
}
```

Além do procedimento de atualização, é necessário incluir a chamada a esse procedimento, dentro da função `loop()`, de forma periódica. Além disso, a função `loop()` também recebe uma nova forma de consulta, caso ocorra erro no WiFi ou no acesso ao programa PHP. Para esses casos, a consulta será local, ao arquivo já gravado. A função `loop()` deve ficar assim:

```
unsigned long anterior_tarefa = 0;

void loop() {
    char code_RFID[20] = "";
```

```
    int httpCode = 0;

    if ((millis() - anterior_tarefa > 100000) && (WiFi.status() ==
WL_CONNECTED)) {
        atualiza_local();
        anterior_tarefa = millis();
    }

    if (read_RFID(code_RFID)) { // Verificar se o cartão foi lido

        if(WiFi.status() == WL_CONNECTED) { // Verifica a conexão WiFi
            HTTPClient http;
            Serial.print("[HTTP] Início...\n");
            // Configura URL, define cabeçalhos e monta o método POST
com a TAG
            http.begin("http://sergiool.esy.es/consult.php");
            http.addHeader("Content-Type", "application/x-www-form-
urlencoded");
            httpCode = http.POST(String("tag=") + code_RFID);
            // httpCode será negativo em caso de erro
            if(httpCode > 0) {
                // Obteve resposta do servidor
                Serial.printf("[HTTP] POST ... código: %d\n",
httpCode);
                // Achou o arquivo
                if(httpCode == HTTP_CODE_OK) { // Conectou, atualiza o
arquivo
                    String payload = http.getString();
                    Serial.println(payload);
                    if (payload == "OK") { // Localizou, libera o
acesso
                        digitalWrite(D0, HIGH);
                        delay(100); // Tempo suficiente para acionar o
acesso
                        digitalWrite(D0, LOW);
                    }
                }
            } else
                Serial.printf("[HTTP] POST... código: %d\n", httpCode);
            http.end();
```

```
        }
        if ((WiFi.status() != WL_CONNECTED) || (httpCode !=
HTTP_CODE_OK)) {
            // Houve erro na consulta ao banco
            File f = SPIFFS.open("/f.txt", "r");
            String local = f.readString();
            f.close();
            if (local.indexOf(code_RFID) > 0) { // Localizou no arquivo
                Serial.printf("Acesso local liberado: %s\n",
local.c_str());
                digitalWrite(D0, HIGH);
                delay(100); // Tempo suficiente para acionar o acesso
                digitalWrite(D0, LOW);
            }
        }
    }
}
```

Também é necessário incluir o arquivo *FS.h* na lista de includes.

A função `millis()` foi usada para verificar o tempo de atualização, sendo considerado o intervalo de 100 segundos para isso. Poderia ser maior, apesar de ser uma função bem rápida. Para localizar o código no arquivo, lido em uma string, foi usada a função `indexOf()` para localização de uma substring em uma string. Ela retorna -1 se não localizar.

Outra demanda necessária é formatar o sistema de arquivos, algo que deve ser feito na primeira utilização. Como isso é feito apenas uma vez, será incluído na função de inicialização `setup()`. A formatação só será executada se o arquivo ainda não existir no sistema de arquivos. Caso o arquivo já esteja presente, a formatação não ocorrerá mais em futuras reinicializações. O código inserido é mostrado a seguir.

```
SPIFFS.begin(); // Inicia o sistema de arquivos
if (!SPIFFS.exists("/f.txt")) {
    Serial.println("Não localizou o arquivo. Formatando ...");
    SPIFFS.format();
}
```

O programa, dessa forma, funcionará para liberar o acesso das tags já cadastradas, mesmo que o acesso à rede WiFi, à internet ou ao servidor

utilizado esteja indisponível, o que é essencial em um sistema de acesso, visto que nem sempre é possível garantir a disponibilidade nesse tipo de sistema. O arquivo completo está no repositório disponível com o livro.

CAPÍTULO 13

Interfaces com smartphones

Com a popularização dos smartphones, seus aplicativos se tornaram interfaces interessantes para Internet das Coisas, e podem, ainda, ser associados às suas múltiplas funcionalidades para criar aplicações extremamente úteis. Este capítulo apresenta o ambiente de desenvolvimento Android Studio, usado para criar aplicativos para smartphones Android na linguagem Java. Uma aplicação exemplo é apresentada com a integração do protocolo MQTT à linguagem Java para servir de interface alternativa ao controle de irrigação apresentado no capítulo 11.

A **evolução e a popularização dos smartphones** os tornaram interfaces ideais para diversos tipos de aplicações, incluindo aplicações já populares em outros dispositivos, como acesso a redes sociais, fotografia e vídeo; e até mesmo aplicações antes inexistentes como monitoramento de condições de trânsito para escolha da melhor rota, rastreamento, compartilhamento etc. No cenário de IoT, smartphones parecem ser as interfaces ideais para acesso aos diversos tipos de dispositivos.

Os smartphones já estão **integrados às redes IoT**, pois têm as mesmas formas de comunicação – WiFi, Bluetooth, NFC – e contêm um grande conjunto de sensores: GPS, acelerômetro, magnetômetro, câmera etc. Logo, é esperado que as aplicações de IoT considerem os smartphones de forma totalmente integrada. A maior limitação é o custo dos smartphones, que torna impeditivo sua utilização de forma dedicada em aplicações de baixo valor agregado. No entanto, considerando sua disponibilidade como dispositivo pessoal para a maioria dos usuários, é possível conceber diversas funções para os smartphones no cenário de IoT.

Os smartphones são uma **evolução dos telefones celulares**. Inicialmente grandes e pesados, à medida que perderam em tamanho, ganharam em

poder de processamento, software e recursos. Alguns recursos foram estratégicos nesse desenvolvimento, como as telas sensíveis a toque capacitivas, que aboliram definitivamente os teclados. Os primeiros sistemas proprietários dos fabricantes deram espaço aos sistemas Android, IOS e Windows Phone. E os equipamentos ganharam processadores com recursos próximos aos dos computadores pessoais, à medida que seu preço e a disposição dos usuários em gastar subiram consideravelmente.

Entre os sistemas, o **Android** é um sistema aberto, baseado no Linux e desenvolvido pelo Google, que o recheia de seus aplicativos gratuitos, mas repletos de anúncios comerciais. O **IOS** é fechado, restrito aos celulares produzidos pela Apple. E o **Windows Phone**, também fechado, tenta ganhar espaço entre vários fabricantes que podem licenciar o software da Microsoft para seus aparelhos, mas tem a popularidade muito menor em relação ao Android ou ao IOS.

Há várias formas de utilizar os smartphones para comunicar e como interface para dispositivos IoT. A primeira e mais imediata é como um **navegador web** portátil. Todas as aplicações web aqui apresentadas para interfacear com IoT podem ser acessadas de smartphones, usando seus navegadores, os quais estão disponíveis em grande variedade para cada sistema. A maior vantagem de usar uma interface web é que ela pode ser compatível, ao mesmo tempo, com vários tipos de dispositivos e sistemas, como computadores pessoais, smartphones e tablets.

A utilização de **interface web para smartphones** é uma tendência, especialmente com a linguagem HTML 5. Repleta de recursos para a criação de aplicações, a linguagem HTML 5 possibilita a criação de aplicativos completos para smartphones, com a vantagem de executarem em qualquer sistema que suporte essa linguagem. Basicamente, todos os sistemas modernos a suportam. A Intel criou a ferramenta Intel XDK para desenvolvimento de aplicativos móveis baseados em HTML 5. A proposta é portabilidade para qualquer tipo de sistema.

O desenvolvimento de **aplicativos específicos** para executar em smartphones, em especial **Android e IOS**, é ainda uma tarefa com uma demanda enorme. O momento do desenvolvimento de software apresenta uma migração das aplicações desktop e web para o ambiente móvel.

Apesar de existir uma tendência à utilização de HTML 5, alguns recursos e interfaces dos smartphones ainda são mais facilmente acessados pelos aplicativos específicos.

Para este curso, será considerado o desenvolvimento para aplicativo Android, por ser aberto e mais acessível. Diversas ferramentas de desenvolvimento estão disponíveis, sendo a mais recomendada o **Android Studio**, ferramenta desenvolvida pelo Google a partir da IDE IntelliJ IDEA. Outras ferramentas estão disponíveis, como o MIT Inventor, que é programado em blocos, ou o Eclipse ADT, que, assim como o Android Studio, é uma IDE Java com muitos recursos.

13.1 Android Studio

O Android Studio é o IDE oficial de desenvolvimento para Android. Criado pelo Google, ele tem um ambiente de criação de interfaces do tipo arraste e solte que facilita a criação das aplicações. Utiliza a linguagem Java para definição dos algoritmos e das estruturas de dados e a linguagem XML para interfaces e configurações. Cada tela desenvolvida no ambiente gera um arquivo em linguagem XML que é compilado junto à aplicação. O resultado é um arquivo .apk que pode ser disponibilizado no Google Play, loja oficial de aplicativos para Android.

O site oficial para download é *https://developer.android.com/studio/index.html*. É preciso paciência; além de baixar 1,6 GB, o processo demora um bom tempo para instalar e configurar o sistema. Vai demorar mais ainda se não estiver disponível um smartphone Android para o desenvolvimento. O ambiente contém um emulador para testar as aplicações, mas é muito pesado e não vai executar em tempo satisfatório em computadores com poder de processamento típico, como um sistema com processador Core i3 e 4 GB de RAM.

A linguagem de desenvolvimento é **Java**. Linguagem gratuita, desenvolvida pela Oracle, que comprou a Sun, empresa que iniciou a linguagem Java. A linguagem é totalmente orientada a objetos e executa sobre uma máquina virtual, conceito que possibilita que um programa já compilado em *bytecodes* execute em qualquer plataforma de hardware ou software. Na prática, isso não funciona tão bem para plataformas com hardware muito

distinto, como um smartphone e um computador pessoal, mas já resolve boa parte dos problemas de compatibilidade entre diferentes sistemas e plataformas.

A figura 13.1 apresenta o ambiente de desenvolvimento do Android Studio. Observa-se a interface de arrastar e soltar componentes na tela. O resultado é um arquivo XML, com as definições obtidas a partir dessa interface.

Figura 13.1 – Android Studio.

O resultado da inserção dos dois botões e um campo de texto, como visto acima, leva ao seguinte código em XML:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.sergio.myapplication.MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:id="@+id/textView1" />
    <Button
        android:text="Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginLeft="60dp"
        android:layout_marginStart="60dp"
        android:layout_marginTop="25dp"
        android:id="@+id/button1"
        android:onClick="Clique (MainActivity)" />
    <Button
        android:text="Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/button1"
        android:layout_toRightOf="@+id/textView1"
        android:layout_toEndOf="@+id/textView1"
        android:layout_marginTop="82dp"
        android:id="@+id/button2"
        android:onClick="Clique (MainActivity)" />
</RelativeLayout>
```

Esse código pode ser editado, atualizando automaticamente na interface. O sentimento sempre é de que: veja o que você está fazendo enquanto está fazendo.

A lógica do programa é definida em arquivos em Java. Os componentes e

eventos são manipulados em classes associadas a interfaces. As interfaces estão associadas a **Activity ou atividade**. Uma *Activity* está sempre associada a um arquivo XML, que define sua interface, e a um arquivo Java, que define sua lógica. Como a linguagem é totalmente orientada a objetos, a *Activity* é associada a uma classe e terá a seguinte sintaxe:

```
public class MainActivity extends AppCompatActivity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

O método `onCreate` é executado sempre ao iniciar a atividade. Nesse método, é necessário incluir as inicializações e associações gerais, como variáveis a componentes incluídos na interface. Para acessar propriedades dos componentes, é preciso declarar variáveis para referenciá-los nessa classe e localizá-los com a função `findViewById()`. Essa função retorna uma referência ao componente. Para associá-la a uma variável do tipo do componente, é preciso identificar seu tipo entre parênteses, como pode ser visto no código a seguir.

```
public class MainActivity extends AppCompatActivity {

    TextView t;
    Button b;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        t = (TextView) findViewById(R.id.textView1);
        b = (Button) findViewById(R.id.button1);
    }
}
```

O tratamento de eventos também faz parte dessa classe. Para isso é necessário definir um método para tratar esse evento. Esse método deve ser do tipo `public void`, e receber como parâmetro um objeto da classe `View`. O código a seguir ilustra um método para tratar um evento, o método `Clique()`.

```
public class MainActivity extends AppCompatActivity {

    TextView t;
    Button b;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        t = (TextView) findViewById(R.id.textView1);
        b = (Button) findViewById(R.id.button1);
    }

    public void Clique(View v) {
        t.setText("Ola, Mundo!");
    }
}
```

Para associar esse evento ao clique do botão é preciso acessar essa propriedade na interface de criação de telas, ou no código XML correspondente.

Também é possível fazer essa associação diretamente no código, usando a função `setOnClickListener()`. Ela pode ser atribuída no método `onCreate()` da Activity. Essa opção parece ser mais estável. Definir no XML pode gerar erros durante a execução do aplicativo.

```
public class MainActivity extends AppCompatActivity {

    TextView t;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        t = (TextView) findViewById(R.id.textView1);
        final Button button = (Button) findViewById(R.id.button1);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                t.setText("Ola, Mundo!");
                // Efetuar ação no clique do botão
```

```
            }
        });

    }
}
```

13.2 App com protocolo MQTT

O protocolo MQTT tem uma implementação em Java disponibilizada pelo Eclipse, com suporte também ao Android Studio. É chamada *Paho Android Service*. Sua utilização é simples, e o Android Studio é capaz de buscar as bibliotecas e disponibilizá-las.

O aplicativo a ser apresentado será mais uma interface para a aplicação de controle de irrigação, apresentada no capítulo anterior, agora diretamente no aplicativo para smartphone. Contará com campos para o envio da Umidade Máxima e Mínima e um campo para apresentar a Umidade Atual, enviada pelo módulo de irrigação. O aplicativo mantém-se inscrito na comunicação de bomba ligada e solenoide ligada, e essas informações serão apresentadas como notificações do aplicativo, com vibração e toque.

O primeiro passo é informar as bibliotecas que serão utilizadas. Para isso é necessário inserir as linhas mostradas a seguir para o arquivo de configuração do Gradle, um gerenciador de bibliotecas e dependências para a geração dos aplicativos no Android Studio. As linhas devem ser inseridas ao final do arquivo *build.gradle (Module: app)*.

```
repositories {
maven {
        url "https://repo.eclipse.org/content/repositories/paho-releases/"
}

dependencies {
    compile('org.eclipse.paho:org.eclipse.paho.android.service:1.0.2')
{
        exclude module: 'support-v4'
    }
}
}
```

Para acessar o arquivo de configuração do Gradle, clique duas vezes sobre *Gradle Scripts* na área **Projects** e depois clique em **Structure**, na barra lateral à esquerda. Em seguida, clique duas vezes em *build.gradle* e edite o arquivo que será apresentado, inserindo as linhas ao final. Após inserir as linhas ao final do arquivo, é necessário sincronizar, clicando no link que aparece acima do arquivo sendo editado. As bibliotecas serão baixadas e estarão disponíveis para utilização. A figura 13.2 mostra a atualização do Gradle.

Figura 13.2 – Atualização das bibliotecas do Gradle.

Também é necessário editar o arquivo *AndroidManifest.xml*, que define as características da aplicação. Esse arquivo é usado pelo Google Play e pelo sistema Android para definir parâmetros de execução da aplicação. A primeira inserção serve para indicar o serviço de rede utilizado. Essas linhas são adicionadas à tag `<application>`. Seguem:

```
<service android:name="org.eclipse.paho.android.service.MqttService" >
</service>
```

Também é necessário dizer que tipo de recurso do sistema será necessário para que o usuário dê permissão para sua utilização na sua instalação. Para isso devem ser incluídas as seguintes linhas à tag `<manifest>`:

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.VIBRATE" />
```

Essas permissões são necessárias para que o aplicativo possa identificar o estado da rede, do telefone e da internet e realizar a notificação com vibração.

13.3 Interface

A interface do aplicativo terá dois componentes visuais para apresentar a umidade corrente e o nível da caixa-d'água, além de dois campos de texto para obter valores de configuração para a umidade máxima e mínima e um botão para publicar as mensagens de umidade máxima e mínima. Além disso, contará apenas com rótulos para esses campos. A figura 13.3 apresenta a interface.

Figura 13.3 – Interface do aplicativo.

A umidade corrente será apresentada em um componente do tipo *ProgressBar*, uma barra de progresso que apresenta graficamente uma fração. Esse componente apresentará a fração da umidade, considerando como o valor total a umidade máxima que pode ser lida pelo sensor. Além de ser apresentada graficamente nesse componente, também será

apresentada em modo textual em um componente da classe `TextView`.

A situação da caixa-d'água, indicando se esta se encontra cheia ou vazia, será apresentada graficamente em um componente do tipo `ImageView`. Esse componente apresenta uma imagem que será escolhida entre duas imagens possíveis. Uma imagem representará a caixa cheia e a outra, a caixa vazia. Para que as imagens possam ser acessadas pelo aplicativo, elas devem ser armazenadas na pasta do projeto, subpasta *app\src\main\res\drawable*. As imagens salvas nesta pasta são acessadas diretamente pelo Android Studio.

A publicação das mensagens referentes à configuração de umidade máxima e mínima de acionamento dos módulos de irrigação será baseada em dois campos de entrada de texto, do tipo `EditText`, e um botão, cujo evento de clique será usado para a publicação das mensagens.

A organização dos elementos na tela requer um componente de leiaute. Para isso foi usado o `GridLayout`, um componente que apresenta uma tabela na qual os demais componentes podem ser inseridos. Cada componente deve ser indicado em qual linha e coluna se encontra, por meio das propriedades `layout_column` e `layout_row`. Para melhor integração dos componentes, o número de linhas e colunas deve ser indicado nas propriedades do GridLayout: `columnCount` e `rowCount`.

Essa interface é salva em um código XML. Esse código pode ser facilmente copiado para outro projeto, visto que é padrão.

O código XML é este:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.sergio.appirrigamqtt.MainActivity">

    <GridLayout
        android:layout_width="368dp"
        android:layout_height="495dp"
        android:columnCount="2"
```

```
android:rowCount="6"
tools:layout_editor_absoluteY="8dp"
tools:layout_editor_absoluteX="8dp">

<ProgressBar
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"
    android:id="@+id/progressBar"
    android:max="100"
    android:layout_column="0"
    android:layout_gravity="top|center_horizontal"
    android:layout_row="0"
    android:layout_width="116dp"
    android:layout_height="37dp"
    android:progress="50"
    android:indeterminate="false" />
<ImageView
    android:layout_width="131dp"
    android:layout_height="126dp"
    android:id="@+id/imageView"
    android:layout_column="1"
    android:layout_gravity="top|center_horizontal"
    android:layout_row="0"
    app:srcCompat="@drawable/caixavazia" />
<TextView
    android:text="50%"
    android:layout_width="84dp"
    android:layout_height="63dp"
    android:layout_column="0"
    android:layout_row="0"
    android:layout_gravity="bottom|center_horizontal"
    android:id="@+id/textViewUm"
    android:textSize="36sp" />
<TextView
    android:text="Umidade"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView"
    android:layout_below="@+id/progressBar"
    android:layout_alignLeft="@+id/progressBar"
```

```
    android:layout_alignStart="@+id/progressBar"
    android:layout_column="0"
    android:layout_gravity="top|center_horizontal"
    android:layout_row="1" />
<TextView
    android:text="Caixa d'água"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:id="@+id/textView5"
    android:layout_column="1"
    android:layout_gravity="top|center_horizontal"
    android:layout_row="1" />
<TextView
    android:text="Umidade Máxima"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView2"
    android:layout_marginTop="16dp"
    android:layout_below="@+id/editTextMax"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_column="0"
    android:layout_gravity="left|top"
    android:layout_row="3" />
<EditText
    android:layout_width="189dp"
    android:layout_height="wrap_content"
    android:inputType="textPersonName"
    android:ems="10"
    android:id="@+id/editTextMin"
    android:layout_alignBaseline="@+id/textView2"
    android:layout_alignBottom="@+id/textView2"
    android:layout_alignLeft="@+id/editTextMax"
    android:layout_alignStart="@+id/editTextMax"
    android:layout_column="1"
    android:layout_gravity="left|top"
```

```
    android:layout_row="3"/>
<EditText
    android:layout_width="189dp"
    android:layout_height="wrap_content"
    android:inputType="textPersonName"
    android:ems="10"
    android:id="@+id/editTextMax"
    android:layout_alignParentTop="true"
    android:layout_toRightOf="@+id/textView"
    android:layout_toEndOf="@+id/textView"
    android:layout_column="1"
    android:layout_gravity="left|top"
    android:layout_row="4" />
<TextView
    android:text="Umidade Mínima:"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView3"
    android:layout_alignBaseline="@+id/editTextMax"
    android:layout_alignBottom="@+id/editTextMax"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_column="0"
    android:layout_gravity="left|top"
    android:layout_row="4" />
<Button
    android:text="Publicar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button1"
    android:onClick="onClickPub"
    android:layout_marginTop="27dp"
    android:layout_below="@+id/editTextMax"
    android:layout_toLeftOf="@+id/textView5"
    android:layout_toStartOf="@+id/textView5"
    android:layout_marginRight="13dp"
    android:layout_marginEnd="13dp"
    android:layout_column="0"
    android:layout_gravity="top|center_horizontal"
```

```
            android:layout_row="5"
            android:layout_columnSpan="2" />
    </GridLayout>
</android.support.constraint.ConstraintLayout>
```

13.4 Organização do código

O código da classe `Activity` principal contém vários métodos, sendo:

1. *callback*, para tratamento de eventos;
2. `onCreate()`, para inicialização das variáveis e da interface;
3. `initMQTT()`, para inicialização dos objetos de conexão;
4. `connectMQTT()`, para realizar a conexão ao broker;
5. `startNotifications()`, para inicialização dos objetos necessários para o envio de notificações;
6. `onClickPub()`, para tratamento do clique do botão, gerando a publicação de mensagens de umidade máxima e mínima para ativação da irrigação.

```
public class MainActivity extends AppCompatActivity {

    private MqttCallback ClientCallBack = new MqttCallback() {...}

    private IMqttActionListener MqttCallBackApp = new
IMqttActionListener() {...}

    // Método de inicialização da Activity
    protected void onCreate(Bundle savedInstanceState) {...}

    // Método de inicialização do cliente MQTT
    private void initMQTT() {...}

    // Inicialização do MQTT e conexão inicial
    private void connectMQTT() {...}

    // Cria as classes necessárias para notificações: Intent,
PendingIntent
    // e acessa o mNotificationManager
```

```
    private void startNotifications() {...}

    // Assina as mensagens MQTT desejadas
    public void subscribeMQTT() {...}

     // Trata o clique do botão, publicando as mensagens
    public void onClickPub(View v) {...} // Evento disparado no clique
do botão
}
```

Os seguintes atributos pertencem à classe correspondente à *Activity* principal:

```
   // Declarações de Referências de Componentes de Interface
    private EditText UMaxima, UMinima;
    private ProgressBar pB;
    private ImageView iV;
    private TextView uText;

    // Declaração para as mensagens de depuração
    private static final String TAG =
MainActivity.class.getSimpleName();

    // Declarações e inicializações para o MQTT
    private String clientId = MqttClient.generateClientId();
    private MqttAndroidClient client;
    private boolean assinou = false;

    // Declarações e inicializações para as notificações
    private NotificationCompat.Builder mBuilder =
                new NotificationCompat.Builder(this);
    private TaskStackBuilder stackBuilder =
TaskStackBuilder.create(this);
    private NotificationManager mNotificationManager;
    private int cont = 0;
```

Os primeiros atributos correspondem às referências para os componentes de interface. A *String TAG* corresponde ao rótulo de depuração e representará o nome da classe. A seguir, aparecem os objetos necessários para a comunicação MQTT e, por último, os objetos necessários para disparar notificações no sistema Android.

13.5 Funções Callback e chegada de mensagens

Dois tipos de funções *callback*, usadas em resposta a eventos, são usadas na comunicação MQTT Android: *MqttCallback*, para resposta a eventos do cliente MQTT, e *IMqttActionListener*, usada em resposta a ações específicas do MQTT, como estabelecimento de conexão e assinatura de postagens. Para facilitar o entendimento e a modularização do código, a definição das funções *callback* foi feita no início da classe relativa à *Activity*.

A função callback `connectionLost()` será chamada sempre que o aplicativo perder a conexão com o broker. Nesse caso, será iniciada uma nova conexão, para que o aplicativo se mantenha sempre conectado.

A função callback `messageArrived()` indica a chegada de uma mensagem de algum tópico inscrito. Nesse ponto, serão tratadas as mensagens recebidas, sendo: mensagens de umidade, do módulo de irrigação, atualizando o componente `ProgressBar` correspondente; mensagens de nível de água, atualizando a imagem correspondente no componente `ImageView`; e as demais mensagens gerando notificações do sistema Android.

As notificações serão configuradas com o nome do tópico, o texto da mensagem, um aviso sonoro padrão do sistema e uma vibração do aparelho, se o recurso estiver disponível. O tempo de vibração é definido no método `setVibrate()`, que recebe um vetor como parâmetro, em que é definido o intervalo no qual o aparelho vibra e fica sem vibrar, em sequência.

O método callback `onSucess()` será acionado quando a conexão com o servidor for estabelecida, bem como quando as assinaturas de tópicos forem efetivadas. É necessário que se aguarde a conclusão do processo de conexão, notificado por esse método, para iniciar o processo de assinaturas. Logo, o método `subscribeMQTT()` será chamado a partir da confirmação da conexão bem-sucedida nesse método. Para evitar que essa assinatura se repita de forma indevida, uma variável booleana chamada `assinou` será usada, sendo verdadeira quando a função já tiver sido chamada. Ela se tornará falsa se a conexão cair, pois isso vai requerer a assinatura dos tópicos, novamente, quando houver uma nova conexão ao broker.

O código é mostrado a seguir.

```
private MqttCallback ClientCallBack = new MqttCallback() {
        @Override
        public void connectionLost(Throwable cause) {
            Log.d(TAG,"Perda de conexão... Reconectando...");
            connectMQTT();
            assinou = false;
        }
        @Override
        public void messageArrived(String topic, MqttMessage message)
                 throws Exception {
            String msg = new String(message.getPayload());
            Log.d(TAG, topic + ": " + msg);
            if (topic.equals("/Umidade")) { // Apresentada graficamente
                int Umidade = Integer.parseInt(msg);
                pB.setProgress(Umidade);
                uText.setText(msg);
            } else if (topic.equals("/NivelAgua")) {
                nText.setText(msg);
                if (msg.equals("Alto")) {
                    iV.setImageResource(R.drawable.caixacheia);
                } else {
                    iV.setImageResource(R.drawable.caixavazia);
                }
            } else {
                // Mensagem publicada
                int mId = 99;
                mBuilder.setSmallIcon(R.drawable.caixacheia)
                      .setContentTitle("Mensagem recebida")
                      .setContentText(topic + ": " + msg)
                      .setVibrate(new long[]{150, 300, 150, 600}) //
Para Vibrar
                      .setSound(RingtoneManager.getDefaultUri(
                                              RingtoneManager.TYPE_NOTIFIC
                // Para apitar
                // Executa a notificação
                mNotificationManager.notify(mId, mBuilder.build());
            }
```

```
        }
        @Override
        public void deliveryComplete(IMqttDeliveryToken token) {
            Log.d(TAG,"Entregue!");
        }
    };
    private IMqttActionListener MqttCallBackApp = new
IMqttActionListener() {
        @Override
        public void onSuccess(IMqttToken asyncActionToken) {
            Log.d(TAG, "onSuccess");
            if (!assinou) {
                subscribeMQTT();
                assinou = true;
            }
        }
        @Override
        public void onFailure(IMqttToken asyncActionToken,
                              Throwable exception) {
            // Problemas na assinatura
            Log.d(TAG, "onFailure");
        }
    };
```

A função `Log.d()` é usada para depuração. Ela envia mensagens para o console do Android Studio se o modo de depuração estiver ligado. Para isso o objeto `TAG` deve ser declarado no início da classe, como segue:

```
// Declaração para as mensagens de depuração
    private static final String TAG =
MainActivity.class.getSimpleName();
```

13.6 Criando a conexão

O estabelecimento da conexão com o broker foi dividido em dois métodos: `initMQTT()` e `connectMQTT()`. O método `initMQTT()` só será chamado uma vez, na inicialização da *Activity*. Já o método `connectMQTT()` é chamado sempre que houver perda de conexão, visando reconectar imediatamente.

Os métodos são mostrados a seguir. Os objetos necessários devem ser

declarados no início da classe e também são listados:

```
// Declarações e inicializações para o MQTT
private String clientId = MqttClient.generateClientId();
private MqttAndroidClient client;

// Método de incialização do cliente MQTT
private void initMQTT() {
    client=new MqttAndroidClient(this.getApplicationContext(),
            "tcp://broker.shiftr.io:1883", clientId);
    client.setCallback(ClientCallBack);
}
// Inicialização do MQTT e conexão inicial
private void connectMQTT() {
    try {
        MqttConnectOptions options = new MqttConnectOptions();
        options.setUserName("cap_iot_");
        options.setPassword("iotiot256".toCharArray());
        IMqttToken token = client.connect(options);
        token.setActionCallback(MqttCallBackApp);
    } catch (MqttException e) {
        e.printStackTrace();
    }
}
```

O tratamento de erros `try {} catch(exception) {}` é necessário para registrar e tratar possíveis erros de conexão.

Duas funções callback são usadas para responder a eventos de sucesso e falha de conexão e apenas registrarão mensagens para a depuração. Elas chamam a função `Log.d()`.

13.7 Assinatura de tópicos

O método `subscribeMQTT()` é responsável por assinar os tópicos necessários, a saber, `/Umidade`, `/Bomba`, `/Solenoide`, `/NivelAgua`. Após assiná-los, ele atribui os métodos callback, que serão chamados em resposta aos pedidos de assinatura.

Esse método deve ser chamado apenas após a resposta de conexão bem-sucedida, o que é percebido pelo evento tratado pela função callback

`onSucess()`. Essa mesma função, no entanto, também responde aos eventos de assinatura de tópicos. Logo, para evitar repetição da assinatura, uma variável booleana é usada para que esse método seja chamado apenas uma vez. Em caso de queda de conexão, a variável é reinicializada, possibilitando que o método seja chamado novamente.

Segue o código da função. O tratamento de exceções também se faz necessário.

```
// Assina as mensagens MQTT desejadas
 public void subscribeMQTT() {
     int qos = 1;
     try {
         if (!client.isConnected()) {
             connectMQTT();
         }
         IMqttToken subTokenU = client.subscribe("/Umidade", qos);
         subTokenU.setActionCallback(MqttCallBackApp);
         IMqttToken subTokenB = client.subscribe("/Bomba", qos);
         subTokenB.setActionCallback(MqttCallBackApp);
         IMqttToken subTokenS = client.subscribe("/Solenoide", qos);
         subTokenS.setActionCallback(MqttCallBackApp);
         IMqttToken subTokenN = client.subscribe("/NivelAgua", qos);
         subTokenN.setActionCallback(MqttCallBackApp);
     } catch (MqttException e) {
         e.printStackTrace();
     }
 }
```

13.8 Envio de notificações

O envio de mensagens para os tópicos `/Bomba` e `/Solenoide`, que indicam o acionamento do sistema de irrigação, vai gerar notificações, incluindo a execução de um som e a vibração do smartphone. Esses eventos são tratados na função callback de recepção de mensagens. Para que isso seja possível, algumas inicializações são necessárias. Essas inicializações serão realizadas no método `startNotifications()`, cujo código é mostrado a seguir:

```
// Cria as classes necessárias para notificações: Intent,
```

```
PendingIntent;
  // e acessa o mNotificationManager
    private void startNotifications() {
        Intent resultIntent = new Intent(this, MainActivity.class);
        TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
        stackBuilder.addParentStack(MainActivity.class);
        stackBuilder.addNextIntent(resultIntent);
        PendingIntent resultPendingIntent =
                stackBuilder.getPendingIntent(
                        0,
                        PendingIntent.FLAG_UPDATE_CURRENT
                );
        mBuilder.setContentIntent(resultPendingIntent);
        mNotificationManager =
                (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
    }
```

Os objetos das classes `Intent`, `TaskStackBuilder` e `PendingIntent` são necessários para que a aplicação possa enviar notificações. Elas configuram o objeto `mBuilder`, que é definido como atributo da classe. Para que as notificações possam funcionar, os seguintes atributos devem ser definidos na classe:

```
// Declarações e inicializações para as notificações
private NotificationCompat.Builder mBuilder = new
NotificationCompat.Builder(this);
private NotificationManager mNotificationManager;
```

13.9 Publicação de mensagens

A publicação de mensagens será realizada em resposta ao evento de clicar o botão. O método `onClickPub()` foi criado com esse objetivo. O código segue abaixo. O protocolo MQTT exige que as mensagens sejam enviadas no formato UTF-8, razão pela qual a conversão é feita a partir do valor obtido no campo EditText.

```
public void onClickPub(View v) { // Evento disparado no Clique do botão
      try {
           if (!client.isConnected()) {
```

```
                connectMQTT();
            }
            client.publish("/UmidadeMaxima",
            new MqttMessage(UMaxima.getText().toString().getBytes("UTF-
8")));
            client.publish("/UmidadeMinima",
            new MqttMessage(UMinima.getText().toString().getBytes("UTF-
8")));
        } catch (UnsupportedEncodingException | MqttException e) {
            e.printStackTrace();
        }
    }
```

Todo o código-fonte está disponível no repositório do livro no GitHub. É importante ficar atento às mudanças nas versões Android, bem como às modificações do Android Studio. A versão apresentada foi compilada no Android Studio 2.3.1 e gerada para executar em versões do Android a partir da versão 4.0. Atualizações são constantes, e é importante identificar o que há de novo e as diferenças de cada versão.

Aplicativos são, atualmente, uma excelente interface para Internet das Coisas. Nesse cenário, o Android predomina, com um enorme percentual dos dispositivos em uso. Versões para iPhone e Windows Phone também são interessantes para abranger um número maior de potenciais usuários. Inúmeros recursos adicionais para esses sistemas, bem como para o Android, estão disponíveis na literatura e podem agregar novas funcionalidades às aplicações, como, por exemplo, recursos de localização dos usuários, entre outros.

CAPÍTULO 14

Raspberry Pi e outras placas Linux

Módulos com o sistema operacional Linux, como o Raspberry Pi, entre outros, têm algumas vantagens sobre os módulos que executam sistemas monolíticos, como aqueles baseados em Arduino: têm maior poder computacional; contêm mais interfaces, como HDMI e USB; contêm drivers para um grande número de periféricos; e podem integrar toda a base de software desenvolvida para Linux. No entanto, chegam a ser dez vezes mais caros e requerem maior consumo de energia. Devem ser reservados para funções específicas, que não podem ser atendidas pelos módulos de baixo custo. Este capítulo apresenta o funcionamento geral desses módulos e alguns exemplos implementados na linguagem Python, que possibilitam aplicações que não são concebíveis para módulos mais simples.

A popularização do **sistema operacional Linux** e de sua versão de código aberto incentivou vários desenvolvedores a migrar o código do sistema para outras famílias de microprocessadores, além da família Intel PC x86. Arquiteturas como PowerPC, ARM e MIPS receberam versões do Linux, com boa parte de seus aplicativos e controladores, tudo o que não seria específico para uma ou outra arquitetura. Com o tempo, até microprocessadores com poder computacional inferior e memória limitada receberam versões do sistema.

A **arquitetura ARM** foi uma dessas arquiteturas premiadas com o sistema Linux. E a escalabilidade dessa arquitetura, disponível em diversas configurações e versões, além do baixo consumo de energia, a tornou a arquitetura preferida para smartphones e tablets. Esse foi um dos motivos que fizeram com que o Google investisse no Linux como base para o sistema Android. No entanto não foi só o Android que se desenvolveu para a arquitetura ARM. O sistema Linux continuou a se desenvolver,

especialmente para placas e módulos com propósitos de implementações específicas e dedicadas, como centrais telefônicas, equipamentos para automação, entre outros.

Aproveitando a arquitetura já existente, e barateada pelo desenvolvimento dos smartphones, e, ainda, o sistema operacional Linux já estável para essa arquitetura, uma fundação sem fins lucrativos, conhecida por ***The Raspberry Pi Foundation***, lançou as primeiras versões do Raspberry Pi A+ e B+, como opções para computadores de baixo custo, focadas principalmente em aplicações de inclusão digital. O projeto incluía também portas GPIO, possivelmente por já estarem disponíveis no microprocessador usado. Essas portas, associadas ao baixo custo da placa e à disponibilidade de software, tornaram o Raspberry Pi um módulo adequado para várias aplicações em sistemas embarcados, em especial, aos primórdios dos conceitos de Internet das Coisas.

Inicialmente lançada nos **modelos** A+ e B+, as placas Raspberry Pi contam hoje com duas atualizações do modelo B+, versões 2 e 3, além de um novo modelo Raspberry Pi Zero, um modelo mais simples, de baixo custo, focado em aplicações IoT, com preço oficial de 5 dólares, preço este que ainda não é possível encontrar, dada a grande procura por esse modelo. Os modelos variam de configuração, mas todos têm um processador da família ARM, capaz de executar o sistema operacional Linux, além de saída HDMI ou micro HDMI, coprocessador gráfico capaz de codificar vídeo H.264 *full HD*, entradas USB, áudio, e praticamente tudo necessário para que possa funcionar como um pequeno computador. Apenas a versão 3 tem rede WiFi, que pode ser agregada às demais com um adaptador USB.

As placas Raspberry Pi, e as similares, têm várias **vantagens** para projetos de IoT: i) contêm processadores potentes, capazes de executar softwares e algoritmos com altas demandas de processamento; ii) se utilizam de um sistema operacional completo, bem conhecido e com ampla disponibilidade de software; iii) englobam interfaces e *device drivers* para um grande número de periféricos, desde teclado, monitores, adaptadores de rede, câmera, áudio, leitores biométricos e outros mais. Como **desvantagem** é possível considerar o custo, acima dos módulos com o ESP8266, especialmente levando em conta o adaptador WiFi, essencial para projetos IoT, além de consumir mais energia e de suas soluções

abrangerem maior complexidade. O ambiente Linux pode ser mais complexo para novatos, que precisam conhecer bem o sistema operacional para configurá-lo.

14.1 Sistema operacional Linux

O sistema operacional Linux é um sistema aberto, com código disponível e amplamente trabalhado por uma grande comunidade ao redor do mundo. Conta com um **núcleo** (kernel) comum, mantido pelo criador do sistema, o finlandês **Linus Torvalds**. A principal característica do Linux é exatamente esse núcleo comum, gerenciado pela equipe do seu criador. Sobre esse núcleo, foi agregado um grande número de aplicativos e utilitários, vindos de um projeto chamado GNU. Esse projeto já estava em grande desenvolvimento quando Linus fez o seu kernel, mas lhe faltava um núcleo. O kernel de Linus Torvalds se adequou facilmente ao projeto, completando esse sistema aberto que rapidamente se popularizou. O sistema é desenvolvido sobre o padrão POSIX, que determina as interfaces das APIs e a organização geral do sistema, facilitando a portabilidade de código e aplicações entre os sistemas operacionais do modelo Unix, dos quais o Linux faz parte. Dessa forma, um grande número de softwares pode ser portado e compilado para o Linux, que o torna um sistema amplo e completo.

O **sistema de arquivos** do Linux é organizado em uma árvore, com pastas e subpastas, assim como a maioria dos sistemas operacionais. No entanto, não conta com o conceito de unidade de disco. Cada unidade deve ser mapeada em uma pasta dentro da mesma árvore de pastas e subpastas. Algumas pastas contêm informações especiais: */etc* contém as configurações do sistema; */proc* contém registradores e informações de operação do sistema, como informações sobre os processos, adaptadores de redes, protocolos e outros dispositivos; */dev* apresenta os diversos dispositivos mapeados em arquivos virtuais; */usr* contém os programas executáveis comuns a todos os usuários; */var* contém arquivos de dados e informações necessárias aos programas, como dados de servidores; */home* contém as pastas pessoais dos usuários cadastrados.

A **interface gráfica** do Linux não é um componente obrigatório do sistema.

Ela foi incluída a partir da implementação livre *XFree86* do sistema *X Windows System*, um sistema de interface gráfica baseada em janelas utilizada nos sistemas Unix. Logo, é possível executar o Linux sem interface gráfica, e isso é ideal para as aplicações IoT. A interface gráfica consome muitos recursos do sistema, especialmente memória e processamento, além de exigir um processador gráfico ou tomar boa parte do processador principal. Nas aplicações IoT sem interface gráfica, os componentes de interface não precisam ser instalados, o que reduz também a demanda de armazenamento. O mesmo ocorre em servidores como os servidores web, bancos de dados etc.

O Linux pode ser executado e configurado a partir de uma **interface texto**, que pode ser enviada para uma interface serial, dispensando até mesmo a necessidade de uma saída gráfica VGA ou similar. Isso reduz bastante o custo de uma placa capaz de executar o Linux, embora a maioria das placas atuais inclua a interface gráfica para atender também à perspectiva de funcionar como um computador pessoal. Uma das prerrogativas do Raspberry Pi que o tornaram popular foi exatamente a possibilidade de utilizá-lo como um computador de baixo custo. No entanto a utilização do Raspberry Pi como dispositivo IoT sem interface gráfica fica bem mais ágil e proporciona respostas mais rápidas se a interface gráfica estiver desabilitada. Por isso, o domínio dos comandos e do sistema de arquivos do Linux é fundamental para os desenvolvedores de dispositivos IoT com o Raspberry Pi.

O gerenciamento de tarefas do Linux é **preemptivo**, ou seja, os processos ocupam o processador apenas por uma fatia de tempo, depois do qual o controle é repassado a outras tarefas. Dessa forma, não é preciso se preocupar com o tempo de resposta que os programas podem obter de outros equipamentos ou dispositivos. Se um processo parar de responder porque está aguardando uma resposta, não há problemas para o Linux, ao contrário do que ocorre com as aplicações desenvolvidas no ambiente Arduino. Assim, o desenvolvedor tem mais flexibilidade para desenvolver suas tarefas. Os processos podem, ainda, ser visualizados, finalizados e ter, até mesmo, sua prioridade alterada. Versões do kernel Linux com características de **tempo real** estão disponíveis para o Raspberry Pi, garantindo a execução de tarefas prioritárias em tempos curtos, para

atender às aplicações que demandam o atendimento em tempo real.

14.2 Instalação e configuração Raspbian

Dentre as muitas distribuições de Linux, inclusive para o Raspberry Pi, uma delas se tornou oficial, por ser suportada diretamente pelo site do Raspberry Pi. A distribuição **Raspbian** pode ser baixada diretamente do site oficial (*www.raspberrypi.org*) em duas versões: uma completa, que inclui interface gráfica, e outra *lite*, sem interface gráfica, mais leve, que pode ser usada para os dispositivos IoT. A vantagem de utilizar a versão gráfica é que ela pode ser usada como ambiente de desenvolvimento para as aplicações aqui apresentadas, aproveitando o propósito de ser, também, um computador pessoal. Logo, é interessante pegar as duas versões do Raspbian e usar uma para o desenvolvimento e a outra para a produção.

Após baixar a versão completa do Raspbian, é preciso gravar um cartão de memória microSD com o sistema. O tamanho mínimo do cartão microSD é de 2 GB para a versão *Lite,* sendo recomendado 4 GB. Para tanto, utiliza-se o software Win32DiskImager, software gratuito que pode ser obtido em *http://sourceforge.net/projects/win32diskimager*. Talvez seja necessário executá-lo como administrador, dependendo da versão do Windows utilizada. Após a instalação e a execução do Win32DiskImager, a tela mostrada na figura 14.1 será apresentada. Para Linux ou Mac, há outros procedimentos a seguir que podem ser encontrados no site oficial do Raspberry.

Figura 14.1 – Interface do Win32DiskImager para gravação do Raspbian.

É preciso indicar o local onde o arquivo da imagem foi salvo e a unidade em que está inserido o cartão microSD. Para usar o cartão microSD é possível usar entradas para cartão que já estão presentes na maioria dos computadores, ou adaptadores para a interface USB. A todos é associada uma letra de unidade. O cartão deve estar limpo e formatado, e o arquivo de imagem deve ser descompactado antes de ser gravado no cartão microSD.

Após gravar a imagem, ela deve ser inserida no Raspberry Pi, que, ligado à fonte e a um monitor HDMI, deve iniciar sua interface gráfica. Ligue um teclado e um mouse nas portas USB disponíveis. Se o Raspberry não tiver todas as portas USB, é possível usar um hub USB. Caso a versão da Raspberry não tenha interface WiFi, também é possível ligar um adaptador WiFi usando um hub USB ou diretamente à porta, se disponível.

É importante ter atenção à questão da energia. Além de observar o consumo, que pode aumentar com a inclusão de periféricos, também é preciso dar atenção à capacidade do regulador de tensão do Raspberry para alimentar os periféricos. Mesmo com a utilização de uma fonte de alta corrente, acima de 2 A, é possível que o regulador de tensão do Raspberry Pi não suporte a inclusão de periféricos que tenham consumo um pouco mais alto, como um HD externo. Nesse caso, é possível usar um hub USB com alimentação própria para ligar dispositivos de consumo maior ao Raspberry Pi.

O nome inicial de usuário para acesso ao Raspberry é `pi` e a senha é `raspberry`. Recomenda-se alterar a senha e até o usuário no primeiro acesso. Não se deve manter os dados de acesso padrão, visto que são facilmente descobertos por pessoas interessadas no mau funcionamento ou curiosos que desejam obter dados sigilosos.

Uma vez instalado e em funcionamento, é possível selecionar quais pacotes de software são necessários no Raspberry. A versão completa disponível para download contém diversos pacotes que não devem ser necessários para uma aplicação IoT, incluindo *Libreoffice*, navegadores para acesso a Internet, entre outros. A versão *Lite* é mais indicada para aplicações IoT, visto que não tem todos esses pacotes, além da interface gráfica.

É possível instalar aplicativos e servidores necessários. Para isso, o comando `sudo apt-get install` *<nome do software>* pode ser usado em um terminal texto. Se não souber exatamente o nome do software, é possível localizar facilmente em pesquisa na internet. Há milhares de aplicativos, servidores, plataformas de desenvolvimento, entre outros, disponíveis para instalação automática. Para remover algum programa instalado, o comando é `sudo apt-get remove` *<nome do software>*. Há também uma interface gráfica para o gerenciamento dos pacotes de software instalados, chamado Synaptic. Para utilizar o Synaptic, basta fazer sua instalação com o comando `sudo apt-get install synaptic` e depois acessá-lo pelo menu **Preferences**.

14.3 Desenvolvimento com o Raspberry Pi

Há grandes vantagens no desenvolvimento de programas para o Raspberry Pi: i) é possível desenvolver no próprio Raspberry, pois ele tem todas as ferramentas de desenvolvimento necessárias; ii) é possível usar uma infinidade de linguagens e bibliotecas disponíveis; iii) é possível integrar com ferramentas disponíveis para esse ambiente, como servidores, softwares matemáticos e estatísticos. Assim, o ambiente de desenvolvimento é tão poderoso quanto um PC, além das vantagens do baixo custo, tanto de aquisição quanto de manutenção e consumo de energia.

Entre os ambientes de programação mais populares para o Raspberry está o ambiente IDLE para a linguagem Python, uma linguagem interpretada, de desenvolvimento rápido e com grande número de bibliotecas e ferramentas para análise matemática e estatística, processamento gráfico e de imagens, integração a servidores e tudo o que se espera para uma linguagem de programação. Os programas desenvolvidos em Python podem, ainda, ser portados com facilidade para um grande número de plataformas e sistemas, incluindo Windows e Linux para PC, além de ambientes móveis como smartphones e tablets, Android e IOS.

Para acessar as portas GPIO do Raspberry Pi com o Python está disponível a biblioteca RPi.GPIO. Para acessá-la, basta incluí-la no início do programa:

```
import RPi.GPIO as GPIO
```

Em seguida, é preciso definir qual modelo de numeração das portas GPIO será seguido. É possível seguir a numeração da placa, ou a numeração do CI utilizado. Para isso é necessário usar o comando `GPIO.setmode()`, com o parâmetro `GPIO.BOARD` para usar a numeração da placa, ou `GPIO.BCM` para usar a numeração do CI.

A configuração das portas como entradas ou saídas é feita pelo comando `GPIO.setup()`, usando como parâmetros o número da porta e o tipo: `GPIO.INPUT` ou `GPIO.OUTPUT`. Para a leitura das portas, a função é `GPIO.input()`, e para a escrita, `GPIO.output()`.

Um programa para piscar um LED na porta GPIO 10 do Raspberry seria bem simples, como pode ser visto no código a seguir:

```
import RPi.GPIO as GPIO
import time
led = 10
GPIO.setmode(GPIO.BOARD)
GPIO.setup(led, GPIO.OUT)
while True:
    GPIO.output(led, GPIO.LOW)
    time.sleep(1)
    GPIO.output(led, GPIO.HIGH)
    time.sleep(1)
```

A função `time.sleep()` serve para que o programa aguarde um tempo, em segundos. Diferente da função `delay()` do Arduino, a função `time.sleep()` é uma espera não ocupada, visto que o sistema é preemptivo. Logo, vários programas como esse poderiam executar de forma concorrente, sem que um interfira no funcionamento do outro. Essa é uma das grandes vantagens do Raspberry sobre plataformas que usam programas monolíticos, como o Arduino.

A linguagem Python é não tipada, ou seja, não é necessário definir nem declarar o tipo das variáveis antes de usá-las. Também não é necessário o uso de ponto e vírgula após os comandos ou as chaves para abrir e fechar laços. A quebra de linha já indica o fim de comando e promove a sua execução. A indentação define os limites dos laços, condicionais e funções. Isso obriga o desenvolvimento de programas mais limpos e com indentação

correta. Mas é preciso cuidado. Erros na indentação vão implicar em erros de lógica ou de execução.

É possível conceber um número ilimitado de aplicações para IoT com o Raspberry Pi, pois há um grande número de periféricos que podem ser ligados ao Raspberry Pi, como câmeras, áudio, impressoras etc. A porta USB possibilita a ligação de um grande número de dispositivos, grande parte deles já com suporte, em software, para o sistema Linux, sendo facilmente integrados aos programas. Além disso, adaptadores Bluetooth aumentam o leque de possibilidades de periféricos que podem ser acionados pelo Raspberry Pi.

14.4 Comunicação em tempo real com o ESP8266

Há diversas aplicações e motivos para comunicar um Raspberry Pi com um ESP8266: aplicações com módulos heterogêneos, usando um Raspberry Pi como módulo servidor central e os módulos com o ESP8266 como periféricos; aplicações que demandem recursos exclusivos de ambos, como a porta analógica do ESP8266, e o servidor web Apache do Raspberry; ou ainda usando o ESP8266 para expandir as portas do Raspberry. Enfim, são várias demandas de comunicação entre esses módulos.

A forma mais simples de comunicação entre eles é pela rede TCP/IP. O ESP8266 possui rede WiFi. O Raspberry também tem interface de rede, Ethernet, que suporta TCP/IP, além de poder receber um adaptador WiFi. Usando a rede TCP/IP, a comunicação pode ser estabelecida em vários protocolos e padrões: soquetes TCP ou UDP, requisições HTTP, MQTT etc. O problema é que a comunicação sobre redes TCP/IP não tem parâmetros de qualidade de serviço, ou seja, não é possível dar nenhuma garantia da comunicação realizada na rede, como o tempo máximo para entregar uma informação, nem mesmo garantia de que ela vá chegar a seu destino. E isso torna inviável sua utilização em aplicações críticas e de tempo real, nas quais a falta de uma informação pode comprometer o sistema.

Aplicações de tempo real envolvem exigências muito específicas em relação aos sistemas computacionais. O Raspberry Pi não foi criado para aplicações de tempo real. Apesar de o Raspberry Pi suportar um sistema

operacional completo, seu sistema mais utilizado, o Raspbian, não suporta operações de tempo real. Isso significa que não é possível, por exemplo, controlar um servomotor, cuja operação depende de controlar sinais da ordem de milissegundos. É possível substituir o kernel do Raspbian por um kernel de tempo real, mas não é uma operação trivial. Modificações do kernel implicam em perda de compatibilidade em atualizações, bem como possíveis instabilidades no funcionamento.

O mesmo não ocorre com placas sem sistema operacional, com código monolítico, como é o caso da maioria das placas Arduino, incluindo os módulos com o ESP8266. É possível deixar o processador monitorando um determinado sinal durante todo o seu tempo de operação, o que garante respostas rápidas para as aplicações de tempo real. Assim, nas aplicações de Internet das Coisas que envolvam requisitos de tempo real, é melhor deixar essa tarefa para o ESP8266. Caso seja necessário sua comunicação com o Raspberry Pi, ou até mesmo com outros módulos microprocessados, também é possível usar uma rede de comunicação que tenha tempo de resposta atendendo a requisitos de tempo real. Ou seja, em vez de utilizar a rede TCP/IP, é possível usar um barramento, como o I2C, para obter maior velocidade e garantia de tempo de resposta na comunicação entre os módulos.

Tanto o Raspberry Pi quanto o ESP8266 suportam o protocolo de comunicação I2C e têm bibliotecas que possibilitam sua comunicação de forma simples e direta. O protocolo é simples e utiliza apenas três fios para a conexão física, o que pode incluir vários dispositivos nos mesmos três pinos, usando o mesmo barramento. Um dispositivo deve ser o mestre da comunicação, e os demais, escravos. A comunicação só é permitida entre um mestre e um escravo. Cada escravo tem um endereço de 7 bits, que é usado pelo mestre para indicar com qual escravo vai se comunicar. É possível ter mais de um mestre num barramento, mas só um deles pode estar ativo. Do contrário, haverá colisão.

Será mostrado, a seguir, um código exemplo de comunicação entre o Raspberry Pi e um módulo com o ESP8266, usando I2C. O exemplo será apenas ilustrativo. O Raspberry Pi enviará um número inteiro ao ESP8266, que responderá com o dobro desse número. O Raspberry Pi atuará como mestre, e o ESP8266, como escravo, visto que só é possível um mestre, mas

vários escravos, e as aplicações típicas tendem a ter um Raspberry e vários ESP8266, em função do preço e dos recursos de cada um. O código para o Raspberry será escrito em Python, como no exemplo anterior, e o código para o ESP8266 será feito sobre a plataforma Arduino, podendo ser compilado na IDE Arduino ou na IDE Eclipse.

Inicialmente, será apresentado o código para o módulo ESP8266. A biblioteca que contém as funções para acesso à comunicação I2C é a *Wire.h*. A função `Wire.begin()` inicia o barramento e deve receber o endereço a ser atribuído, caso atue no modo escravo. Se nenhum endereço for informado, a comunicação será iniciada no modo mestre.

O sistema desse programa é baseado nas duas funções callback `recebeDados()` e `enviaDados()`. Isso ocorre porque somente o mestre pode iniciar a comunicação. A função `recebeDados()` trata os dados enviados pelo mestre, e a função `enviaDados()` responde ao mestre, caso ele requisite dados. O escravo não pode enviar dados para o mestre sem que lhe seja solicitado.

```
#include <Wire.h>

#define ENDERECO 0x20
int numero = 0;

void setup() {
    // Inicia o I2C como escravo
    Wire.begin(ENDERECO);

    // Escravo apenas trata eventos pelas funções callback
    Wire.onReceive(recebeDados);
    Wire.onRequest(enviaDados);
}

void loop() {
    delay(100);
}

// Callback para recepção de dados
void recebeDados(int cont) {
```

```
        while(Wire.available()) {
            numero = Wire.read();
         }
    }

    // Callback para envio de dados a partir da requisição do mestre
    void enviaDados() {
        Wire.write(numero*2);
    }
```

Do lado do Raspberry Pi, é preciso iniciar configurando o suporte à comunicação I2C. Inicialmente, é preciso tirar o módulo responsável pela comunicação I2C da *blacklist*, que é uma lista de protocolos bloqueados no ambiente Linux. Para isso é necessário alterar o arquivo */etc/modprobe.d/raspi-blacklist.conf*. Use o editor de texto de sua preferência. Se não conhecer nenhum, pode usar o Nano, editor bem simples que resolve o problema. É importante usar o comando que dá acesso de superusuário, `sudo`, para editar os arquivos de acesso restrito.

```
sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

A linha que indica o módulo responsável pela comunicação I2C deve ser retirada, ou transformada em comentário, incluindo um caracter # no início da linha. O módulo responsável pela comunicação I2C é o módulo `i2c-bcm2708`. Localize esse módulo no arquivo e exclua a linha correspondente, ou insira o caracter # no início da linha. Assim:

```
#blacklist i2c-bcm2708
```

Também será necessário incluir o carregamento do módulo `i2c-dev` no arquivo */etc/modules*. Use novamente o editor:

```
sudo nano /etc/modules
```

E inclua `i2c-dev` ao fim do arquivo.

A seguir, instale as bibliotecas necessárias para o uso do I2C no Raspberry e suporte no Python com o comando `apt-get`:

```
sudo apt-get install i2c-tools python-smbus
```

Para que o usuário-padrão do Raspberry Pi tenha acesso à comunicação I2C sem a necessidade de usar o superusuário, é preciso adicioná-lo ao

grupo `i2c`, com o comando:

```
sudo adduser pi i2c
```

Em seguida, é preciso reiniciar o Raspberry Pi.

```
sudo reboot
```

Após a reinicialização, deve ser possível localizar o arquivo correspondente à porta de comunicação I2C, com o comando `ls`:

```
ls /dev/i2c*
```

Que deverá apresentar como saída algo como:

```
/dev/i2c-0
```

Esse arquivo será usado para acessar a porta de comunicação I2C.

Após fazer as ligações elétricas com o módulo ESP8266 e ligá-lo, execute o comando `i2cdetect`, como mostrado a seguir.

```
i2cdetect -y 1
```

e verá uma saída como esta:

```
     0 1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: 20 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Que indica endereços nos quais foram detectados dispositivos escravos. Nesse caso, apenas o ESP8266 é mostrado no endereço 20, em hexadecimal. O Raspberry já está pronto para rodar o programa como mestre, para se comunicar com o ESP8266.

O programa para essa comunicação, escrito em Python, é apresentado a seguir.

```
import smbus
import time
# Inicializa a comunicação
bus = smbus.SMBus(0)
```

```
# Define o endereço no qual se encontra o ESP8266 - escravo
escravo = 0x20

while True:
    numero = input("Digite um número: ")
    if numero:
    # Função para enviar dado para o escravo
        bus.write_byte(endereco, numero)
        print ("Foi enviado: ", numero)
        # Espera um segundo
        time.sleep(1)
        # Função para requisitar e receber dado do escravo
        dobro = bus.read_byte(endereco)
        print ("ESP respondeu com ", dobro)
```

O programa é bem simples. Ele lê um número e o envia pela porta I2C para o endereço do ESP8266. Em seguida, aguarda 1 segundo para requisitar a resposta que, recebida, é impressa na tela.

Esse exemplo serve apenas para ilustrar a comunicação I2C entre o Raspberry Pi e o módulo com o ESP8266. Diversas aplicações poderiam ser implementadas usando essa comunicação. Sua demanda é imediata para aplicações de tempo real, visto que ela é rápida e determinística, ou seja, é confiável e é possível saber o tempo exato em que vai ocorrer, com atraso mínimo. Para todas as outras aplicações a comunicação pela rede TCP/IP deve ser mais interessante.

A ligação física entre os módulos é muito simples. Além do terra (GND), apenas mais dois fios são necessários, ligando os pinos 3 (SCL) e 5 (SDA) do Raspberry Pi aos pinos D1 (SCL) e D2 (SDA) do ESP8266, respectivamente. Ambos trabalham na tensão de 3,3 volts e funcionam sem problemas com tensão ou conexão.

Esse exemplo serviu, basicamente, para mostrar mais uma forma de comunicação entre dispositivos IoT, além da comunicação TCP/IP. A maior vantagem, nesse caso, é a confiabilidade da comunicação, bem como o atendimento aos requisitos de tempo real, que podem ser necessários em várias aplicações.

CAPÍTULO 15

Porteiro eletrônico pelo smartphone

Neste capítulo, será apresentada a **implementação de um porteiro eletrônico**, com câmera opcional, que pode ser atendido por um smartphone, ou até vários smartphones. O sistema será implementado sobre um Raspberry Pi, pode ser a Raspberry Pi Zero ou qualquer outra. Como periféricos, será necessário um botão de acionamento, além de microfone USB. Opcionalmente, é possível ligar a câmera do Raspberry Pi, ou outra câmera com suporte no sistema, para enviar também o vídeo de quem está chamando no interfone.

15.1 Apresentando o porteiro eletrônico

A popularização dos **porteiros eletrônicos** já representou grande avanço no conforto e na segurança das residências, possibilitando identificar e até atender à porta da residência sem a necessidade de abri-la. No entanto, diante da necessidade de mais segurança e conforto, é possível melhorar o processo, possibilitando o atendimento diretamente por um smartphone qualquer de um morador, mesmo que ele não esteja em casa no momento. Isso pode apresentar uma série de vantagens, como identificar e espantar ladrões que estejam verificando se a casa está vazia, resolver problemas que não exijam a presença física do morador, ou até agendar com o visitante um horário em que o morador esteja presente.

Diante do aumento da violência urbana, sempre é preciso se preocupar com a possibilidade de furto a residências. Os ladrões, nesse caso, buscam uma forma de identificar se a residência está vazia antes de entrar. Para isso, o método mais comum é chamar pela campainha ou pelo interfone, que, se não forem respondidos, indicam que a casa está vazia. Se um morador puder atender ao chamado no **porteiro eletrônico pelo seu smartphone**, o ladrão a chamar terá a impressão que a casa não está vazia,

desistindo dos seus planos de furto.

Atender ao porteiro eletrônico pelo smartphone também será útil para **reagendar uma entrega** de encomendas, caso ninguém esteja em casa para recebê-las e até para **responder a solicitações** ou demandas rápidas, como despachar alguém interessado em ler algum livro religioso ou vender uma enciclopédia.

Os componentes necessários são:

- placa Raspberry Pi – Pode ser qualquer modelo: Zero, A+, B+, 2 ou 3. No entanto, como a Raspberry Pi Zero não tem saída de áudio, se ela for escolhida, deverá incluir, também, uma saída de som USB;
- microfone USB;
- botão do tipo push Button;
- resistor 110 KΩ;
- alto-falante com interface P2, ou USB, caso a placa utilizada seja Raspberry Pi Zero;
- adaptador WiFi (opcional), visto que é possível usar a interface Ethernet, já disponível em alguns modelos, ou a interface WiFi, no caso da Raspberry Pi 3;
- hub USB (opcional), caso o número de portas USB disponível não seja suficiente, o que deve acontecer se usar a Raspberry Pi Zero ou A+.

Além dos componentes citados, é necessário ter um smartphone Android, que será usado para atender à chamada pelo interfone.

O funcionamento do interfone será baseado no princípio de VoIP (Voz sobre IP), que utiliza a internet para realizar ligações telefônicas. Para isso será instalado um software de VoIP no Raspberry Pi e outro correspondente no smartphone. A comunicação entre o interfone e o smartphone será feita totalmente pelos protocolos de VoIP.

15.2 Voz sobre IP

Realizar telefonia pela internet, ou VoIP, em tempo real é um princípio utilizado há vários anos, sempre identificado como uma das potencialidades da rede. Vários provedores de serviços de

telecomunicações oferecem essa modalidade de comunicação, que está se tornando popular em aplicativos para smartphones que usam esse recurso, como WhatsApp, Messenger ou Google Hangouts. Na prática, como a rede TCP/IP não garante qualidade de serviço, necessária para a comunicação em tempo real, o serviço ainda deixa um pouco a desejar.

O primeiro padrão proposto e amplamente utilizado para VoIP foi o H.323, proposto pela ITU (International Telecommunication Union – União Internacional de Telecomunicações). Como era um padrão pesado e com muitas camadas e módulos, demorou um pouco para se estabelecer de fato. O padrão seguinte, SIP, proposto pelo IETF (Internet Engineering Task Force – Força Tarefa de Engenharia da Internet), se tornou mais interessante e popular, por ser mais leve e fácil de implementar. Rapidamente surgiram várias implementações de ambos os padrões, algumas livres e gratuitas. Os padrões também se tornaram referência para videoconferências, até mesmo em equipamentos profissionais para salas amplas e auditórios.

O modelo de telefonia, no qual qualquer terminal pode originar uma chamada, não se enquadra no modelo cliente-servidor, usado nas redes TCP/IP, pois no modelo telefônico qualquer terminal pode iniciar uma chamada. Não é possível atribuir funções de cliente e servidor aos terminais telefônicos. Para se adaptar ao modelo cliente-servidor das redes TCP/IP, os padrões VoIP preveem um servidor de conexões que funciona como um “auxílio à lista”, identificando onde estão os programas que atuam como terminais telefônicos, possibilitando sua localização. No padrão H.323, esse servidor é chamado de *Gatekeeper*. No padrão SIP, ele é conhecido apenas como *Proxy SIP*.

A função do Proxy SIP, bem como do *Gatekeeper*, é manter a informação atualizada sobre qual endereço IP está sendo usado por cada usuário registrado em tempo real. Assim, os usuários podem informar apenas uma identificação única para o sistema, como um email, por exemplo. Ao iniciar um aplicativo VoIP, este registra o IP e a identificação do usuário no *Proxy/Gatekeeper*. Quando alguém chama por esse usuário, baseado em sua identificação, o *Proxy/Gatekeeper* informa qual é o endereço IP usado pelo usuário naquele momento, possibilitando o estabelecimento da conexão entre os dois programas que atuam como terminais telefônicos.

Assim, não é preciso manter informações atualizadas sobre os endereços de rede. Basta disponibilizar o identificador único.

Um aplicativo bastante interessante para uso em VoIP é o Linphone, software gratuito, de código aberto, que tem versões para smartphones Android, IOS e Windows Phone, além de versões para computadores, Windows e Linux, incluindo uma versão para o Raspberry Pi, que será utilizada nesse exemplo. Os desenvolvedores do Linphone mantêm também um Proxy SIP, chamado Flexip, também de código aberto e gratuito, que atua como Proxy SIP. Ele pode ser instalado localmente, mas há também um servidor disponível, gratuito, mantido pela mesma equipe no endereço *sip.linphone.org*.

Para o funcionamento do porteiro eletrônico aqui apresentado, o Raspberry Pi executará o Linphone. Os usuários que devem ser chamados, caso o botão do interfone seja acionado, deverão ser cadastrados. Quando um visitante apertar o botão, o aplicativo Linphone fará uma chamada para os usuários cadastrados. Haverá, ainda, a opção de tentar chamar um usuário por vez, de forma sequencial ou prioritária, ou chamar todos os usuários de forma simultânea. A chamada de forma prioritária pode privilegiar os usuários que estiverem na mesma sub-rede TCP/IP do porteiro, favorecendo, assim, quem se encontra na residência.

15.3 Montagem do porteiro eletrônico

O primeiro passo será preparar o Raspberry Pi. Nele serão instalados os principais sistemas. É preciso preparar o sistema operacional Raspbian, instalar o Linphone e o aplicativo que fará o acionamento da abertura do portão. O programa para o acionamento será desenvolvido em Python e se comunicará com um aplicativo Android para fazer o acionamento da abertura do portão.

No Raspberry, a instalação do linphone é feita pelo comando `apt-get`, dessa forma:

```
sudo apt-get install linphone
```

O software Linphone tem interface gráfica como a maioria dos programas gráficos Linux ou Windows. No entanto, para usar no porteiro eletrônico, essa interface será dispensada para que não comprometa o desempenho do

sistema, visto que o desempenho do Raspberry com a interface gráfica não é satisfatório, especialmente nas primeiras versões da placa. Assim, a interface gráfica será deixada de lado e um aplicativo Linphone de linha de comando será usado: `linphonecsh`. Esse aplicativo possibilita fazer chamadas, registrar em um servidor proxy, além de configurar o recebimento automático de chamadas. Ele pode ser usado em qualquer linguagem shell script do Linux. Para manter a continuidade no uso do Python como linguagem de desenvolvimento para o Raspberry Pi, esse comando será embutido em uma aplicação Python.

Neste exemplo, o Raspberry Pi contará com dois programas desenvolvidos em Python: um para controlar a ligação de áudio, possivelmente também com vídeo, a partir do acionamento do botão do porteiro; e outro para controlar a abertura do portão, a partir do acionamento do smartphone. Seria possível escrever um programa único para realizar essas duas tarefas, mas se torna mais simples o seu entendimento se forem dois programas separados. De qualquer forma, seu funcionamento é independente, visto que a abertura pode ser acionada a qualquer momento, até mesmo para acesso regular do morador.

O funcionamento da aplicação Python de controle da ligação VoIP será baseado no estabelecimento de uma chamada com um usuário previamente configurado, sempre que o botão de chamada do interfone for acionado. Outra aplicação Python vai monitorar o envio do comando para acionar a abertura do portão, usando o protocolo MQTT. Dada a sua simplicidade e devido ao fato de já ter sido amplamente explorado ao longo deste livro, o protocolo MQTT é a solução ideal para acionamento do portão, visto que é uma solução na nuvem, sem necessidade de configuração dos equipamentos intermediários de rede.

A seguir, é mostrada a aplicação Python, que vai disparar a comunicação VoIP. Ela é bastante simples. Inicia o Linphone, registra no Proxy SIP e espera pelo acionamento do botão para fazer a chamada VoIP para o endereço do usuário previamente cadastrado. Neste exemplo, o endereço do usuário foi definido estaticamente no código. Poderia estar em um arquivo, banco de dados, servidor web etc. Enfim, essa aplicação pode ser aprimorada de diversas maneiras para flexibilizar a inserção dos usuários responsáveis pelo atendimento.

```
# -*- coding: utf-8 -*-
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(17,GPIO.IN)
#Váriável de controle do botão.
entradaAnterior = False

def chamada_voip():
    saida = subprocess.check_output("linphonecsh status register", \
                            stderr=subprocess.STDOUT, shell=True)
    if saida.find('ERROR') != -1:
        saida2 = subprocess.check_output("linphonecsh init",
            stderr=subprocess.STDOUT, shell=True)
        if saida2 != '':
            print ("Erro na execução do linphonecsh init")
            exit(1)

    if saida == "registered=0" OR saida.find('ERROR') != -1:
        saida = subprocess.check_output("linphonecsh register \
                --host sip.linphone.org \
                --username sergiool \
                --password 7819049455955895", stderr=subprocess.STDOUT,
shell=True)
        if saida != '':
            print ("Erro na execução do linphonecsh register")
            exit(1)
    saida = subprocess.check_output("linphonecsh dial
sergiool5@sip.linphone.org")

while True:
    # Lê o botão do interfone
    entrada = GPIO.input(17)
    # Verifica se já tratou o acionamento, ou seja, se o botão foi
apertado e solto
    if ((not entradaAnterior) and entrada):
        print("Botao pressionado")
        chamada_voip()
    #atualiza a entrada anterior
```

```
        entradaAnterior = entrada
        #pequena pausa não sobrecarregar o processador
        time.sleep(0.05)
```

A chamada VoIP ocorre, basicamente, na chamada à função `subprocess.check_output("linphonecsh dial sergiool5@sip.linphone.org")`. Essa função faz acesso a chamadas do sistema, que, nesse acesso, chamará o programa `linphonecsh`, cliente VoIP, indicando uma chamada para o usuário *sergiool5@sip.linphone.org*. Esse usuário está registrado e iniciado no smartphone no qual será recebida a chamada. Esse smartphone deve ter o aplicativo linphone instalado e em funcionamento, registrado com esse endereço.

Antes, porém, que a chamada VoIP seja iniciada, é preciso iniciar e registrar o aplicativo responsável por gerir o acesso VoIP. A sequência de comandos para o `linphonecsh` é `init`, `register` e `dial`. E essa sequência pode ser realizada na inicialização do sistema. Para esse programa em Python, a função `chamada_voip()` verifica, inicialmente, o status do registro, com a chamada `linphonecsh status register`. Se o retorno dessa chamada indicar `ERROR`, será necessário enviar o comando `init` e, em seguida, `register`. Se indicar `registered=0`, apenas o comando `register` deve ser executado. Em sequência, o comando `dial` pode ser chamado, realizando a chamada para o smartphone.

O programa principal é, basicamente, um laço infinito, que identifica mudança no estado do botão, quando pressionado, para realizar a chamada da função `chamada_voip()`, que realiza as operações com o `linphonecsh`. O estado do botão é armazenado na variável `entradaAnterior`, que tem a função de garantir que só vai fazer a chamada uma vez quando o botão for pressionado, mesmo que permaneça pressionado por um intervalo de tempo significativo.

15.4 Acionamento do portão

O acionamento do portão será feito por um simples comando enviado do smartphone para o Raspberry Pi, que, próximo ao portão, acionará um relé por um pino de GPIO para comandar a sua abertura. Há inúmeros métodos e protocolos que possibilitam fazer esse acionamento, a maioria já

citada neste livro. É possível citar alguns:

- **Conexão direta por socket de rede TCP/IP do smartphone ao Raspberry Pi** – Neste caso, o endereço IP do Raspberry deve ser conhecido *a priori*, o que gera dificuldades adicionais caso o Raspberry esteja em uma rede que não tenha IP fixo, e use NAT para tradução de endereços IP falsos. Isso pode ser superado usando DNS dinâmico e mapeamento do IP e porta no roteador.
- **Acesso direto a um servidor web no Raspberry Pi** – O servidor poderia ter um programa instalado em PHP, ou qualquer outra linguagem, que, ao ser acionado, enviaria o comando para atuar na saída-porta GPIO. Mantém os mesmos problemas da primeira opção, incluindo a necessidade de DNS dinâmico e mapeamento de IP e porta no roteador.
- **Acesso intermediado por um servidor web na nuvem** – Neste caso, tal qual apresentado em capítulos anteriores, o smartphone acionaria um programa web na nuvem, alterando o estado de uma variável para indicar a abertura do portão. O Raspberry deve monitorar constantemente o estado dessa variável, acessando também o mesmo servidor web. Caso o estado se altere, o acionamento do portão deve ser feito. Esse método evita a necessidade de DNS dinâmico e mapeamento de IP e porta no roteador, exigindo uma complexidade a mais na manutenção do servidor web.
- **Protocolo MQTT com broker externo** – Essa opção, já apresentada várias vezes neste livro, utiliza um broker MQTT externo, com várias opções gratuitas e sem necessidade de configuração ou desenvolvimento adicionais, e pode usar qualquer linguagem de programação para seu acionamento. O envio do comando para acionamento pode ser feito por uma página web com um programa JavaScript, armazenado localmente no smartphone, eliminando a necessidade de desenvolvimento do aplicativo. Do lado do Raspberry, um programa em Python (ou qualquer outra linguagem que suporte MQTT) deve assinar o tópico a ser utilizado, aguardando a publicação das mensagens solicitando a abertura do portão.
- **Usando mensagens do programa VoIP** – O padrão SIP prevê o envio de mensagens, e o Linphone suporta o acesso a essas mensagens pela

linguagem Python. A conexão VoIP SIP já estaria estabelecida para a ligação de voz e, opcionalmente, vídeo. Poderia, então, ser também usada para enviar um comando a ser recebido pelo Raspberry Pi e entendido como um comando para abrir o portão. Poderia ser uma senha, ou uma palavra simples, como “ok”, ou qualquer outra.

Dada a simplicidade da implementação, a opção 4 será usada neste exemplo. O protocolo MQTT já foi mostrado na linguagem Java, JavaScript e Arduino. Agora é vez de apresentá-lo em Python. Tão simples como em qualquer outra linguagem, com hospedagem do broker na nuvem, de forma simples e gratuita.

A seguir, será apresentado o programa em Python responsável pelo acionamento do portão. Tal qual o programa que fará a ligação VoIP quando o botão do porteiro eletrônico for acionado, o programa para acionamento do portão deve estar sempre em execução, iniciando com a iniciação do sistema, e se mantendo sempre em execução. Também é importante manter a conexão com o broker MQTT sempre ativa. Assim, é preciso verificar constantemente se ela está ativa, e, caso não esteja, deve ser restabelecida.

```
# -*- coding: utf-8 -*-
import paho.mqtt.client as mqtt
import RPi.GPIO as GPIO
import time
portao = 11
GPIO.setmode(GPIO.BOARD)
GPIO.setup(portao, GPIO.OUT)
def na_conexao(cliente, dados, retorno):
    print('Conectado. Codigo de retorno:' + str(retorno))
# Tópico assinado na conexão. Caso desconecte e seja necessário
reconectar,
# o tópico é assinado novamente
    client.subscribe('/PorteiroEletronico')
# Função callback que trata a chegada de um comando para abrir o portão
def na_publicacao(cliente, dados, msg):
    print ('Topico: '+ msg.topic+ '\nMensagem:' + str(msg.payload))
    if (str(msg.payload) == 'Abre'):
        GPIO.output(portao, GPIO.HIGH)
```

```
        time.sleep(0.05)
        GPIO.output(portao, GPIO.LOW)

client = mqtt.Client()
client.on_connect = na_conexao
client.on_message = na_publicacao
client.username_pw_set('cap_iot_', 'iotiot256')
client.connect('broker.shiftr.io', 1883, 60)
client.loop_forever()
```

O programa tem duas funções do tipo callback que serão usadas para responder aos eventos de conexão e recebimento de mensagens. A função callback para conexão será usada para assinar o tópico utilizado, `/PorteiroEletronico`. A função callback para o recebimento de mensagens será usada para abrir o portão quando a mensagem se referir ao tópico assinado e tiver a palavra "Abre", que será enviada pelo smartphone. Após a definição e a associação das funções callback, o programa tem a parte de conexão e o método `loop_forever()` é usado para manter o programa em execução, recebendo as mensagens MQTT do broker sempre que forem enviadas. Essa função, além de manter o programa ativo, também cuidará da reconexão caso o programa seja desconectado por qualquer motivo. Caso seja reconectado, a função callback para a conexão será chamada, assinando novamente o tópico.

Do outro lado, no smartphone, será usado um arquivo HTML e outro JavaScript, que será incluído no primeiro, que ficarão armazenados e executados localmente no smartphone, usando o conteúdo de MQTT em Javascript, conforme apresentado anteriormente. Esse arquivo pode ter um atalho na tela principal do smartphone, tornando imediato e fácil o seu acionamento. Vai ter apenas um botão que, acionado, enviará o comando para a abertura do portão. Eventualmente, pode ser incluída uma senha ou algum tipo de mecanismo de segurança adicional. Não foi incluído neste momento.

O arquivo HTML apresentado a seguir conta apenas com a inserção das bibliotecas JavaScript, além do código que será executado, e um botão que, ao ser acionado, enviará o comando de abertura do portão.

```
<!DOCTYPE html>
```

```html
<html>
  <head>
    <meta charset="utf-8">
    <title>Porteiro Eletrônico</title>
    <script src="https://code.jquery.com/jquery-2.2.3.min.js"></script>
    <script src="https://assets.shiftr.io/js/mqtt-latest.js"></script>
    <script src="script.js" charset="utf-8"></script>
  </head>
  <body>
    <button id="button">Abrir portão</button>
  </body>
</html>
```

A seguir, o código JavaScript que será executado. A conexão com o broker será realizada na abertura da página. O envio do comando para a abertura será publicado no broker com o comando `publish`, para o tópico `PorteiroEletronico`.

```javascript
$(function() {
    var client =
mqtt.connect('mqtt://cap_iot_:iotiot256@broker.shiftr.io', {
        clientId: 'javascript'
    });

    client.on('connect', function() {
        console.log('Cliente conectado!');
    });

    $('#button').click(function() {
        client.publish('/PorteiroEletronico', 'Abre');
        window.alert('Comando enviado');
    })
})
```

O código JavaScript basicamente conta com a conexão, realizada logo que a página termina de ser carregada, com o comando `mqtt.connect`, e o envio da publicação do comando para a abertura do portão, com o comando `client.publish`. Foi utilizado o framework jQuery, que possibilita um código mais otimizado e condensado. Os testes foram feitos no navegador Google Chrome.

Esse exemplo mostra que é possível criar diversas aplicações com o Raspberry Pi. Em geral, as aplicações que exigem um poder computacional maior, ou a presença de um sistema operacional e/ou aplicativos já existentes, podem ser boas aplicações para o Raspberry Pi. Vale lembrar, ainda, que as versões mais novas já têm suporte a versões do sistema operacional Microsoft Windows, com suporte a diversos outros aplicativos, bibliotecas e ferramentas de desenvolvimento.

Bibliografia

Tanenbaum, A. S. *Redes de Computadores*. 4ª ed. Rio de Janeiro: Campus-Elsevier, 2003.

Espressif Inc. ESP8266 Mesh User Guide. Disponível em: <*https://espressif.com/sites/default/files/documentation/30a-esp8266_mesh_user_guide_en.pdf*>.

Espressif Inc. ESP8266 Low Power Solutions. Disponível em: <*https://espressif.com/sites/default/files/documentation/9b-esp8266_low_power_solutions_en.pdf*>.

Espressif Inc. ESP8266 System Description. Disponível em: <*https://espressif.com/sites/default/files/documentation/0b-esp8266_system_description_en.pdf*>.

Raspberry Pi Foundation. Raspberry Pi Documentation. Disponível em: <*https://www.raspberrypi.org/documentation*>.

Arduino.cc. Arduino Language Reference. Disponível em: <*https://www.arduino.cc/en/Reference/HomePage*>.

Tiba, Chigueru et al. *Atlas Solarimétrico do Brasil*. Disponível em: <*http://www.cresesb.cepel.br/publicacoes/download/Atlas_Solarimetrico_do_*
Recife: Editora Universitária da UFPE, 2000.

Johnson, David B; Maltz, David A. *Dynamic Source Routing in Ad Hoc Wireless Networks*, in *Mobile Computing*. Tomasz Imielinski and Hank Korth (ed.), cap. 5, p. 153-181Dordrecht, the Netherlands, Kluwer Academic Publishers, 1996.

Asaad, Sameh. *Simulation Environment for an Ad-Hoc Wireless Network Running the AODV Routing Algorithm*. Disponível em: <*http://www.ctr.columbia.edu/~angin/e6950/sameh/aodv_final.html*>.